TECHNISCHE UNIVERSITÄT BERLIN

A Web Service Frontend for Archiving and Providing QDA Artifacts

Master's Thesis

Author
Sara Pervana
493197
sara.pervana@campus.tu-berlin.de

Examiners
Prof. Dr. Dirk Riehle
Prof. Dr. Manfred Hauswirth
Prof. Dr. Volker Markl

Technische Universität Berlin, 2026
Institut für Telekommunikationssysteme
Fachgebiet Open Distributed Systems

A Web Service Frontend for Archiving and Providing QDA Artifacts

Master's Thesis

Submitted by:
Sara Pervana
493197
sara.pervana@campus.tu-berlin.de

Technische Universität Berlin
Institut für Telekommunikationssysteme
Fachgebiet Open Distributed Systems

2026

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe. Alle Stellen, die den benutzten Quellen und Hilfsmitteln wörtlich oder inhaltlich entnommen sind, habe ich als solche kenntlich gemacht.

Sofern KI-Tools verwendet wurden, habe ich Produktnamen, Hersteller, die jeweils verwendete Softwareversion und die Art der Nutzung (z. B. sprachliche Überprüfung und Verbesserung der Texte, systematische Recherche) benannt. Für die vorliegende Arbeit wurden folgende KI-Tools für die nachstehenden Zwecke eingesetzt:

- Claude Sonnet 4.6 & 5 (Anthropic): Unterstützung bei Programmieraufgaben, insbesondere zur Fehlersuche und Codeverbesserung.
- ChatGPT GPT-5.5 (OpenAI): Unterstützung bei der sprachlichen Überarbeitung, einschließlich Rechtschreibung, Grammatik, stilistischer Verbesserung sowie der Übersetzung einzelner Wörter und Formulierungen.

Ich verantworte die Auswahl, die Übernahme und sämtliche Ergebnisse des von mir verwendeten KI-generierten Outputs vollumfänglich selbst.

Die Satzung zur Sicherung guter wissenschaftlicher Praxis an der TU Berlin vom 15. Februar 2023* habe ich zur Kenntnis genommen.

Ich erkläre weiterhin, dass ich die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

(Unterschrift) Sara Pervana, Berlin, 13. Juli 2026

*https://www.static.tu.berlin/fileadmin/www/10002457/K3-AMBl/Amtsblatt_2023/Amtliches_Mitteilungsblatt_Nr._16_vom_30.05.2023.pdf

Abstract

Computer-Assisted Qualitative Data Analysis Software (CAQDAS) such as MAXQDA, NVivo, and ATLAS.ti, enable researchers to analyze, organize, and code qualitative research data. The resulting Qualitative Data Analysis (QDA) artifact contains not only primary research data, such as interview transcripts, but also analytical information including annotations, code systems, and other metadata generated throughout the research process. These artifacts are of significant scientific value but are commonly stored in proprietary file formats that limit the interoperability between different CAQDAS tools and complicate their long-term preservation, reuse, and accessibility. Existing research repositories generally treat such artifacts as ordinary files, neither exposing nor supporting their internal structure. QDArchive is an ongoing research project that addresses these challenges by providing a web-based platform for archiving, managing, and sharing QDA artifacts. This platform seeks to support researchers and reviewers by enabling long-term preservation and improving accessibility of qualitative research data while promoting interoperability through standardized formats. This thesis investigates, designs, and implements the first frontend prototype of the QDArchive platform. The developed frontend supports the core functionalities of user authentication, project management, artifact upload and download, and archive exploration. The frontend was implemented using modern web technologies, including Next.js, React, TypeScript, and Material UI, by providing a scalable and maintainable interface for interacting with the underlying QDArchive services. The resulting prototype shows the feasibility of providing an intuitive and accessible interface for managing qualitative research artifacts within QDArchive. Finally, the implemented functionality is evaluated against the initial and evolving system requirements, therefore providing a foundation for future development of the platform and supporting the long-term goal of preserving qualitative research artifacts in an interoperable and reusable manner.

Kurzfassung

Computerunterstützte Software zur qualitativen Datenanalyse (Computer-Assisted Qualitative Data Analysis Software, CAQDAS) wie MAXQDA, NVivo und ATLAS.ti unterstützt Forschende bei der Analyse, Organisation und Kodierung qualitativer Forschungsdaten. Die daraus entstehenden Artefakte der qualitativen Datenanalyse (Qualitative Data Analysis, QDA) enthalten neben primären Forschungsdaten, beispielsweise Interviewtranskripten, auch analytische Strukturen wie Codes, Codesysteme, kodierte Segmente, Memos, Annotationen und weitere im Forschungsprozess erzeugte Metadaten. Diese Artefakte besitzen einen hohen wissenschaftlichen Wert, werden jedoch häufig in proprietären Dateiformaten gespeichert. Dadurch werden die Interoperabilität zwischen verschiedenen CAQDAS-Werkzeugen sowie die langfristige Erhaltung, Zugänglichkeit und Nachnutzung der Artefakte erschwert. Bestehende Forschungsdatenrepositorien behandeln solche Artefakte überwiegend als gewöhnliche Dateien und stellen ihre interne analytische Struktur in der Regel nicht gesondert dar.

QDArchive ist ein laufendes Forschungsprojekt, das diesen Herausforderungen durch die Entwicklung einer webbasierten Plattform für die Archivierung, Konvertierung, Verwaltung, Bereitstellung und kontrollierte Weitergabe von QDA-Artefakten begegnet. Ziel der Plattform ist es, die langfristige Erhaltung qualitativer Forschungsartefakte zu unterstützen, ihre Zugänglichkeit zu verbessern und die Interoperabilität durch standardisierte Austauschformate zu fördern.

Diese Arbeit untersucht, konzipiert, implementiert und evaluiert den ersten Frontend-Prototyp der QDArchive-Plattform. Das entwickelte Frontend unterstützt unter anderem die Benutzerauthentifizierung, die öffentliche Archivsuche, die Konvertierung von QDA-Dateien, die geführte Projekterstellung, die Verwaltung von Projektmetadaten und Dateien, das Ergänzen fehlender referenzierter Dateien, die Versionierung, den Projektdownload sowie eigentümerseitige Sichtbarkeits- und Freigabesteuerungen. Die Implementierung basiert auf Next.js, React, TypeScript, Material UI, TanStack React Query und Supabase-bezogenen Bibliotheken und verwendet eine komponentenorientierte sowie servicebasierte Struktur.

Die Evaluation erfolgte anhand der funktionalen und nicht-funktionalen Anforderungen, manueller Ende-zu-Ende-Szenarien sowie formativer Rückmeldungen von Projektbetreuenden und Mitgliedern des QDArchive-Entwicklungsteams. Die Ergebnisse zeigen, dass insbesondere die Konvertierung, die geführte Projekterstellung, die Verwaltung von Metadaten und Dateien, die Versionierung und der Download im untersuchten Entwicklungsumfeld erfolgreich unterstützt werden. Gleichzeitig bleiben unter anderem die empfangsseitige Darstellung öffentlicher und sicher geteilter Projekte, die direkte Dateiansicht, weiterführende Suchfunktionen sowie Reviewer- und Administrationsoberflächen unvollständig. Der entwickelte Prototyp bildet damit eine funktionale Grundlage für die weitere Entwicklung einer webbasierten Oberfläche zur strukturierten, interoperablen und langfristigen Bereitstellung qualitativer Forschungsartefakte.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Qualitative research plays an important role in disciplines such as sociology, education, psychology, healthcare, human-computer interaction, and information systems. Whereas quantitative research primarily examines numerical measurements and statistical relationships, qualitative research seeks to understand human experiences, social interactions, and meanings through the interpretation of non-numerical material (Creswell & Poth, 2018; Denzin & Lincoln, 2011). Such material may include interview transcripts, field notes, documents, images, audio recordings, videos, and observational records.

The scientific value of a qualitative research project is not limited to the collected material. During analysis, researchers identify relevant passages, assign codes, organize codes into hierarchies, write memos, annotate documents, and establish relationships between different parts of the data. These analytical structures document how researchers move from empirical material toward interpretation and explanation. Consequently, preserving only the original source material would exclude an important part of the knowledge created during the research process.

Since the 1980s, researchers have used Qualitative Data Analysis Software (QDAS), commonly referred to as Computer-Assisted Qualitative Data Analysis Software (CAQDAS), to support these activities (Wolski, 2018). Contemporary tools such as MAXQDA, ATLAS.ti, and NVivo assist researchers in organizing heterogeneous data, coding selected segments, managing code systems, creating annotations and memos, and retrieving analytically relevant material (Paulus et al., 2013; Silver & Lewins, 2014).

The outcome of such work can be understood as a Qualitative Data Analysis (QDA) artifact. A QDA artifact combines primary research data, such as transcripts, images, recordings, and observational documents, with secondary analytical data, such as codes, coded segments, code systems, memos, annotations, and relationships. It therefore represents more than a collection of files: it preserves both the empirical material and parts of the analytical process through which that material was interpreted.

1.2 Problem Statement

QDA artifacts are commonly stored in proprietary file formats defined by the CAQDAS tool in which they were created. Although these formats support continued work within the same software environment, they may limit exchange between tools and create dependencies on individual vendors. As Evers et al. (2020) explain, CAQDAS packages may offer similar visible functionality while differing substantially in their terminology, data models, and internal representation of analytical structures.

These differences create challenges for collaboration, migration, review, and long-term accessibility. Researchers may need to exchange projects with colleagues who use different software, move between institutions with different licences, or revisit projects after older software versions are no longer supported. Reviewers and secondary researchers may also need to inspect the relationship between empirical material and analytical decisions. In these situations, access to the primary files alone is often insufficient because codes, memos, annotations, and coding relationships are required to understand how the analysis was conducted.

The FAIR principles state that research data should be findable, accessible, interoperable, and reusable (Wilkinson et al., 2016). Their application to qualitative research is nevertheless complex. Qualitative data may contain sensitive, personal, or context-dependent information and may be difficult to anonymize without reducing its interpretive value. Archiving and sharing therefore require approaches that balance long-term scientific value with practical and ethical access restrictions (Cliggett, 2013).

Research repositories such as Dataverse, Zenodo, the Open Science Framework, and the Qualitative Data Repository provide important infrastructure for metadata management, persistent identification, versioning, access control, publication, and preservation (Crosas, 2011; European Organization for Nuclear Research (CERN) & OpenAIRE, 2013; Foster & Deardorff, 2017; Qualitative Data Repository, 2024). However, the repositories considered in this thesis primarily manage research outputs as datasets, files, or collections of files. A proprietary CAQDAS project can therefore be deposited and preserved as a binary object while its internal analytical structure remains difficult to inspect without the original application.

The REFI-QDA standard addresses part of this problem by providing exchange formats for transferring analyzed qualitative research projects and codebooks between QDA software packages (Evers et al., 2020). It demonstrates that QDA artifacts can be represented as structured research objects rather than isolated files. Nevertheless, transfer remains dependent on how individual tools represent and support analytical concepts. Some structures may be transformed, interpreted differently, or omitted when a project moves between software environments. Interoperability is therefore not only a file-conversion problem, but also a representational and user-facing problem.

A gap consequently remains between the analytical richness produced in CAQDAS tools and the file-oriented way in which such projects are commonly archived and accessed. Addressing this gap requires system capabilities for storage, conversion, persistence, access control, and preservation, together with a frontend that makes those capabilities understandable and usable. Users must be able to identify project formats, distinguish file roles, recognize missing referenced material, understand visibility conditions, manage versions, and retrieve or share artifacts without needing detailed knowledge of the underlying storage and conversion mechanisms.

1.3 Scope

QDArchive is an ongoing research project that addresses this gap by developing a web-based platform for archiving, managing, finding, converting, downloading, and sharing QDA artifacts. Its broader goal is to support the preservation, review, verification, exchange, and reuse of qualitative research artifacts by making primary research data and associated analytical structures available under appropriate access conditions. QDArchive also seeks to reduce dependency on individual CAQDAS vendors by supporting standardized and convertible representations of QDA projects.

The complete QDArchive system includes responsibilities beyond the frontend. Backend services are responsible for concerns such as data persistence, storage operations, conversion execution, authorization enforcement, file retrieval, and other system-level processes. The frontend developed in this thesis does not itself guarantee long-term preservation, access security, or successful format conversion. Instead, it provides the user-facing layer through which the corresponding system capabilities are presented, initiated, and managed.

The frontend is nevertheless central to the practical usefulness of the platform. Researchers require understandable workflows for creating and managing projects, completing missing referenced files, creating versions, configuring downloads, and controlling visibility. Visitors require low-friction access to archive search and conversion. Future reviewers and administrators require role-appropriate interfaces for inspecting shared artifacts and operating the service. The frontend therefore translates domain-specific and technical system capabilities into navigation structures, pages, forms, status indicators, and guided workflows.

The objective of this thesis is to investigate, design, implement, and evaluate the first frontend prototype of the QDArchive platform. The completed prototype includes:

- a public landing page and legal-information pages;
- email/password registration, authentication, logout, and password recovery;
- a public archive-search interface for authenticated and unauthenticated users;
- a conversion workflow for guest and authenticated users;
- an authenticated dashboard and project list;
- guided project creation;
- authenticated project detail pages;
- project and file metadata management;
- identification and upload of missing referenced primary files;
- management of generic supplementary files;
- project download and versioning workflows;
- basic user-profile settings; and
- responsive layouts for desktop, tablet, and mobile viewport sizes.

Owner-facing visibility and secure-link controls are implemented, while recipient-facing public and secure-link views remain incomplete. File-viewing controls are present, but uploaded files do not currently open reliably. Reviewer and administrator interfaces, named-user sharing, advanced archive filtering, complete licence-aware publication workflows, and several account-management operations also remain outside the completed implementation and are addressed in the evaluation and future-work chapters.

The central research question of this thesis is:

> *How can a frontend prototype for QDArchive support the user-facing workflows required for archiving, managing, finding, and accessing QDA artifacts?*

This question is divided into the following sub-questions:

1. What frontend-specific requirements can be derived from the goals of QDArchive and from the needs of researchers, reviewers, and visitors?
2. How can the main QDArchive workflows be translated into an information architecture, visual design, and user experience design for a web-based frontend?
3. How can modern web technologies be used to implement a maintainable and responsive frontend prototype for QDArchive?

4. To what extent does the implemented frontend prototype satisfy the identified requirements, and which limitations remain for future development?

These questions reflect the frontend- and design-oriented scope of the thesis. The work covers frontend requirements analysis, information architecture, interaction and visual design, implementation, backend integration from the frontend perspective, and evaluation of the resulting prototype. It does not evaluate the internal correctness of the conversion algorithms, long-term preservation infrastructure, backup procedures, or backend authorization mechanisms except where their observable behavior affects the frontend.

QDArchive was developed in an iterative project environment in which interaction-level requirements and available backend capabilities evolved during implementation. Initial requirements were derived from the related work and project documentation and were subsequently refined through implementation experience and supervisor and team feedback. The final consolidated requirement baseline presented in Section 3 is used to evaluate the prototype in Section 6.

1.4 Structure

The remainder of this thesis is structured as follows.

Chapter 2 presents the conceptual and technical background, including qualitative research, QDA artifacts, CAQDAS tools, interoperability challenges, REFI-QDA, research data management, the FAIR principles, and existing research repositories.

Chapter 3 derives the requirements for the QDArchive frontend and defines the relevant user roles, user journeys, functional requirements, and non-functional requirements.

Chapter 4 explains how these requirements were translated into a frontend architecture, information architecture, user experience design, responsive layout strategy, and visual identity.

Chapter 5 describes the implementation of the frontend prototype, including its project structure, data-access mechanisms, public and authenticated workflows, reusable components, backend integration, and implementation limitations.

Chapter 6 evaluates the prototype against the consolidated functional and non-functional requirements through requirements inspection, manual end-to-end workflow validation, implementation inspection, and formative supervisor and team feedback.

Chapter 7 presents a prioritized development roadmap covering recipient-facing access, publication, archive discovery, artifact inspection, reviewer and administrator workflows, accessibility, automated testing, and technical hardening.

Finally, chapter 8 summarizes the work, identifies the principal contributions, answers the central research question, and states the overall conclusions of the thesis.

2 Related Work

This chapter reviews five topics relevant to the QDArchive frontend: qualitative research and QDA artifacts, CAQDAS functionality, interoperability, research data management and FAIR principles, and existing repository infrastructures. The sources include methodological literature, peer-reviewed articles, technical standards, and repository documentation. The review provides a focused foundation for QDArchive rather than a systematic or exhaustive literature review. The selected repositories illustrate different categories of scholarly infrastructure and should not be interpreted as a comprehensive evaluation of all existing systems.

2.1 Qualitative Research and QDA Artifacts

Qualitative research examines human experiences, social interactions, behaviors, and meanings through the interpretation of non-numerical data (Creswell & Poth, 2018; Denzin & Lincoln, 2011). It is widely used in disciplines where contextual understanding is more important than numerical generalization.

Qualitative research material may include interview transcripts, field notes, documents, images, audio recordings, and videos. These data are often heterogeneous, partly structured, and dependent on the context in which they were produced (Creswell & Poth, 2018). Their analysis is typically iterative: researchers examine the material, assign codes to meaningful segments, compare observations, and develop categories and interpretations (Saldaña, 2021). Memos, annotations, code hierarchies, and relationships may also be created to document analytical decisions (Miles et al., 2014).

These analytical products contribute to the transparency, traceability, and confirmability of qualitative research (Silver & Lewins, 2014). Preserving only the original source material would therefore exclude part of the knowledge produced during analysis. The relationship between empirical material, qualitative analysis, and the resulting artifact is illustrated in Figure 2.1.

For this thesis, a qualitative research project contains two complementary categories of information. Primary research data consist of the empirical material collected or produced during the study. Secondary analytical data consist of structures created while interpreting that material, including codes, coded segments, code systems, memos, annotations, categories, and relationships.

Together, these elements form a Qualitative Data Analysis (QDA) artifact. A QDA artifact is therefore more than a document or compressed project file: it combines research evidence with parts of the analytical structure through which that evidence was interpreted. Table 2.1 summarizes its main components.

Understanding QDA artifacts as structured research objects provides the conceptual foundation for the remainder of this chapter. Their preservation and reuse require support beyond conventional file storage.

2.2 Computer-Assisted Qualitative Data Analysis Software (CAQDAS)

Computer-Assisted Qualitative Data Analysis Software (CAQDAS), also called Qualitative Data Analysis Software (QDAS), supports the organization, coding, retrieval, and interpretation of qualitative data. Such tools have developed since the 1980s from lim-

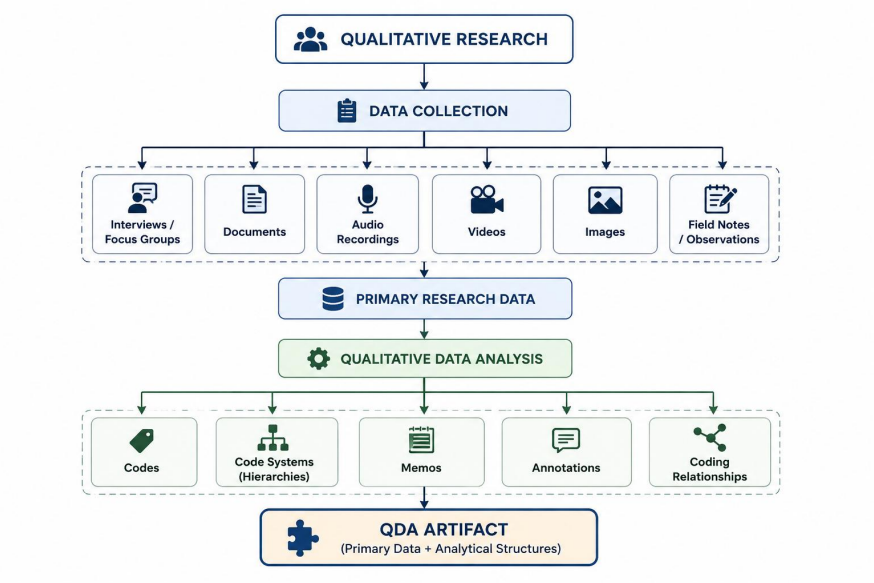


Figure 2.1: Generation of a QDA artifact from qualitative research

Primary research data	**Secondary analytical data**
Interview transcripts	Codes
Audio recordings	Code systems
Videos	Memos
Images	Annotations
Field notes	Coding relationships
Documents	Categories / themes

Table 2.1: Components of a QDA artifact

ited coding-and-retrieval systems into environments for managing heterogeneous research material (Evers et al., 2020; Wolski, 2018).

CAQDAS does not replace the interpretive work of researchers. Instead, it supports the organization, searching, annotation, comparison, and systematic documentation of large or complex datasets (Paulus et al., 2013; Silver & Lewins, 2014). Typical functions include importing material, assigning codes, managing code systems, writing memos, creating annotations, linking analytical elements, retrieving coded content, and visualizing relationships (Di Gregorio & Davidson, 2009; Jackson & Bazeley, 2019; Paulus et al., 2013; Silver & Lewins, 2014).

Table 2.2 summarizes common functions and the artifact elements they produce.

CAQDAS function	**Description**	**Resulting artifact element**
Data import	Import transcripts, documents, images, audio, and video files.	Primary data files
Coding	Assign codes to selected segments of the data.	Coded segments / codings
Code management	Organize codes into hierarchies or thematic structures.	Code system
Memoing	Write analytical notes, reflections, and methodological comments.	Memos
Annotation	Add comments to documents, passages, or selected segments.	Annotations
Linking / visualization	Connect documents, codes, memos, segments, or other objects.	Relationships / links
Querying	Retrieve coded material, patterns, or combinations of elements.	Analytical views / query results

Table 2.2: Typical CAQDAS functions and resulting QDA artifact elements

Widely used packages include MAXQDA, ATLAS.ti, and NVivo. Although they support similar activities, they differ in terminology, functionality, internal data models, and representations of analytical structures. Evers et al. (2020) note that tools may appear similar at the interface level while implementing coding, memoing, or segmentation differently.

CAQDAS project files are therefore not neutral or software-independent containers. They combine source material with software-specific representations of codes, memos, annotations, and relationships. Similar concepts may also use different names, such as code, node, quotation, reference, memo, or annotation.

These differences affect how projects are interpreted, exported, transferred, and archived. QDArchive must consequently support artifacts originating from heterogeneous software environments. Its frontend must present available information and workflows without assuming that users possess the original software or understand its internal model.

2.3 Interoperability of Qualitative Research Data

Interoperability is the ability of systems to exchange information and use the information exchanged (ISO/IEC/IEEE, 2017). For QDA artifacts, this means not only opening a file

in another application but also transferring primary data, codes, coded segments, code systems, memos, annotations, and relationships without losing their intended meaning.

This is difficult because many CAQDAS tools use proprietary formats based on software-specific concepts. A project created in one tool may therefore not be fully readable or editable in another. Corti and Gregory (2011) argue that qualitative data and analytical annotations should remain understandable beyond the lifetime of a particular software environment.

Limited interoperability creates vendor dependency, complicates collaboration across institutions, restricts review and secondary analysis, and introduces preservation risks when software or format support is discontinued. These challenges are summarized in Figure 2.2.

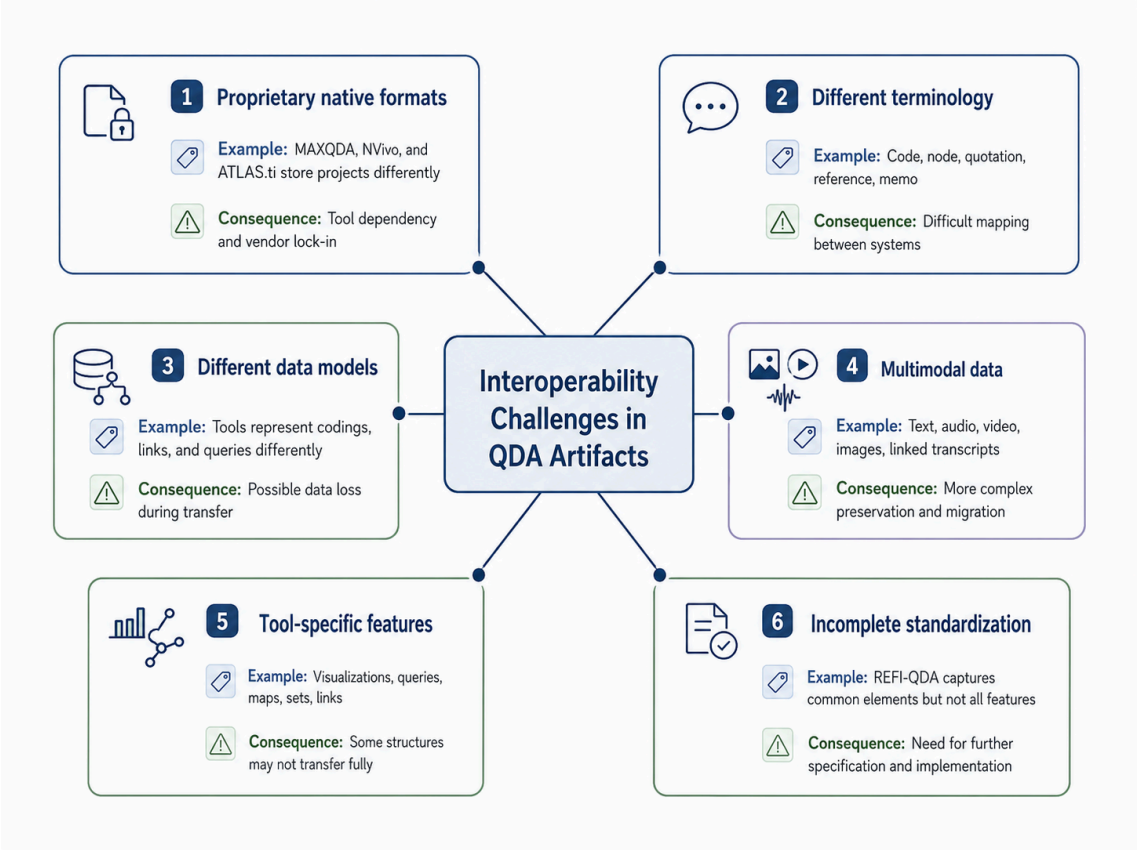


Figure 2.2: Sources of interoperability challenges in QDA artifacts

Several standardization efforts have addressed these problems. Earlier work examined XML-based representations for exchanging and archiving qualitative data (Corti & Gregory, 2011; Muhr, 2000). Hocker et al. (2021) evaluate QualiCO, an ontology for sharing and reusing qualitative coding schemas. Their work shows that interoperability also requires sufficiently rich descriptions of analytical concepts, not only transferable files.

The most directly relevant effort for complete CAQDAS projects is REFI-QDA. It defines a project format for complete analyzed projects and a codebook format for exchanging code systems. These use the QDPX and QDC extensions, respectively (Evers et al., 2020). REFI-QDA supports migration between software packages, collaboration, secondary analysis, and archiving.

However, it does not eliminate all information loss. Differences in software models and supported features may alter or omit coded material, queries, links, and other structures during transfer (Evers et al., 2020). Interoperability therefore depends on both the exchange specification and its implementation by individual tools.

For QDArchive, backend components perform parsing, validation, conversion, and storage, while the frontend communicates supported formats, detected file types, missing referenced files, available targets, and conversion results. Interoperability is therefore both a technical concern and a user-interface concern.

2.4 Research Data Management and FAIR Principles

Research Data Management (RDM) covers the planning, organization, documentation, storage, protection, preservation, sharing, and reuse of research data throughout its lifecycle (Doorn, 2018). In open-science contexts, research data are increasingly treated as outputs that should remain interpretable beyond their original project (Borgman, 2012; Doorn, 2018).

The FAIR principles describe research data as findable, accessible, interoperable, and reusable (Wilkinson et al., 2016). They concern not only files but also the metadata, identifiers, formats, documentation, and access mechanisms required for their use.

Findability requires searchable descriptions, rich metadata, and persistent identifiers. For QDA artifacts, relevant metadata include titles, creators, methods, keywords, data types, formats, licences, and descriptions of analytical content.

Accessibility concerns transparent and appropriately managed retrieval. It does not require unrestricted openness. Qualitative data may contain sensitive or consent-restricted material and may therefore require anonymization or controlled access (Antes et al., 2018; Bishop, 2009). OECD (2021) also connect responsible access with the metadata, workflows, models, and software needed for interpretation.

Interoperability requires documented mechanisms for exchanging and interpreting data across systems. Proprietary CAQDAS formats and tool-specific models make this particularly important for QDA artifacts.

Reusability requires sufficient context and clear conditions for future interpretation. Qualitative material may lose value when separated from methodological documentation, coding decisions, memos, or analytical relationships (Corti, 2007; Karcher et al., 2016).

Applying FAIR to QDA artifacts therefore requires more than preserving a binary file. Their reuse depends on preserving empirical data, analytical structures, metadata, access conditions, and relationships between these elements.

The broader QDArchive system contributes through storage, access enforcement, metadata persistence, format handling, and conversion. The frontend supports these capabilities by allowing users to enter metadata, distinguish access states, search projects, understand versions and file roles, and select conversion or download options. It supports FAIR-oriented workflows but does not by itself guarantee long-term preservation or reuse.

RDM and FAIR therefore explain why QDA artifacts require specialized support, particularly because of their sensitivity, contextual dependence, proprietary formats, and analytical structures.

FAIR principle	Meaning for QDA artifacts	Implication for QDArchive frontend
Findable	Artifacts require project-level metadata, including creators, methods, keywords, formats, licences, and analytical content.	Support searchable metadata, descriptions, tags, summaries, and visibility indicators.
Accessible	Access conditions must be transparent and appropriately controlled.	Communicate access states and provide understandable sharing and download workflows.
Interoperable	Artifacts should remain transferable despite proprietary formats and different internal models.	Communicate supported formats, conversion options, missing files, and download representations.
Reusable	Future users need methodological context, analytical structures, licences, and version information.	Present contextual metadata, versions, artifact contents, and relationships between elements.

Table 2.3: Implications of the FAIR principles for QDA artifacts and QDArchive frontend design

2.5 Existing Research Repositories

Research repositories support the preservation, publication, discovery, citation, and reuse of research outputs. Common services include file storage, metadata management, persistent identifiers, versioning, access control, and curation (Borgman, 2012; Wilkinson et al., 2016).

The repositories reviewed here illustrate different categories of scholarly infrastructure. Zenodo, Dataverse, Figshare, and the Open Science Framework represent general-purpose repositories or workflow platforms. The Qualitative Data Repository and UK Data Service represent services focused more closely on qualitative or social-science data. arXiv is included only as a benchmark for mature public submission, discoverability, citation, and persistent access.

arXiv primarily disseminates scientific manuscripts (arXiv, 2024). Zenodo supports datasets, software, publications, DOI assignment, and versioning (European Organization for Nuclear Research (CERN) & OpenAIRE, 2013). Dataverse provides software for publishing, citing, discovering, and preserving research data (Crosas, 2011). The Open Science Framework combines file sharing, project organization, collaboration, and registrations (Foster & Deardorff, 2017), while Figshare supports the sharing and citation of multiple research-output types (Figshare, 2024).

The Qualitative Data Repository supports curation, contextual documentation, publication, and reuse of qualitative and multi-method data (Qualitative Data Repository, 2024). The UK Data Service provides social-science data collections, deposit guidance, preservation support, and access management (UK Data Service, 2024). These services more directly address consent, anonymization, contextual documentation, and controlled access (Antes et al., 2018; Bishop, 2009; Karcher et al., 2016).

The reviewed services provide important support for metadata, identification, citation, access management, versioning, and curation. However, they generally manage CAQ-

Repository / service	Main focus	Main strengths	Limitation for QDA artifacts
arXiv	Scholarly manuscript archive	Mature public dissemination, discoverability, citation, and persistent access.	Not a research-data repository; included only as a discoverability benchmark.
Zenodo	General-purpose repository	Supports multiple output types, DOI assignment, and versioning.	Does not inspect codes, memos, annotations, or coding relationships.
Dataverse	Research data repository software	Supports publication, metadata, citation, access management, and preservation.	Does not expose the internal analytical structure of QDA formats.
Open Science Framework	Research workflow platform	Combines project organization, sharing, registrations, and collaboration.	Not specialized for inspecting or converting CAQDAS structures.
Figshare	General-purpose repository	Supports sharing and citation of varied research outputs.	Does not provide QDA-specific exploration or interoperability workflows.
Qualitative Data Repository	Qualitative and multi-method repository	Supports curation, contextual documentation, and responsible reuse.	Does not focus specifically on CAQDAS files as structured analytical objects.
UK Data Service	Social-science data service	Provides deposit guidance, preservation, and controlled access.	Does not provide dedicated inspection or conversion of QDA project structures.

Table 2.4: Overview of selected research repositories and archiving services in relation to QDA artifact requirements

DAS projects as deposited files or datasets. Codes, coded segments, memos, annotations, code hierarchies, and relationships may remain inaccessible without the original CAQDAS software.

This limits review, verification, and secondary analysis. Reviewers may need to inspect how evidence was coded, while secondary researchers may require code systems, memos, and analytical relationships to interpret the original work.

QDArchive therefore addresses a more specific requirement than generic repository deposit. QDA artifacts should be represented as structured and versioned research objects whose formats, file roles, missing references, access states, and analytical contents can be exposed through dedicated workflows.

2.6 Research Gap

The preceding sections show that preserving qualitative research requires more than storing source files. QDA artifacts combine empirical material with analytical structures whose value depends on remaining accessible, interpretable, and connected to their context.

CAQDAS tools create and manage these structures but primarily operate within their own software environments. The repositories reviewed in this thesis support storage, metadata, citation, versioning, access control, and preservation, but generally represent CAQDAS projects as files rather than as analytical structures that can be inspected independently.

Consequently, a project file may remain available while its codes, coded segments, memos, annotations, and relationships remain difficult to examine without the original software. This observation applies to the selected services reviewed here and is not intended as a universal claim about all repository infrastructures.

REFI-QDA addresses part of the problem by enabling project exchange, but its limitations show that interoperability also depends on how analytical structures are represented, transformed, and communicated (Evers et al., 2020).

QDArchive addresses this broader problem through archiving, discovery, conversion, controlled access, downloading, and eventual artifact inspection. Backend services provide storage, parsing, conversion, persistence, and access-control enforcement, while the frontend makes these capabilities understandable and operable.

Researchers require workflows for creating projects, entering metadata, uploading files, resolving missing references, creating versions, downloading artifacts, and managing visibility. Visitors require accessible search and conversion, while reviewers and other recipients require read-only access to shared or public projects.

The specific research gap addressed by this thesis is therefore the frontend representation of QDA artifacts. Existing literature provides foundations for qualitative data archiving, interoperability, FAIR-oriented management, and repository services, but comparatively limited guidance on how these domain characteristics should be translated into the information architecture and workflows of a dedicated web archive.

This thesis addresses that gap by deriving frontend requirements from the reviewed literature and project context and by designing, implementing, and evaluating a first QDArchive frontend prototype. The following chapter presents the resulting user roles, journeys, functional requirements, and non-functional requirements.

3 Requirements for QDArchive

This chapter defines the requirements for the QDArchive frontend based on the domain challenges discussed in Chapter 2, the original QDArchive project description, the updated product requirements document (PRD), and the iterative refinement process followed during frontend development. The purpose is not to specify the complete QDArchive backend or preservation infrastructure, but to identify the user-facing requirements through which users can find, convert, create, manage, download, version, and share QDA artifacts.

The requirements are organized around the principal user-facing areas of QDArchive: public access through the landing page and *Find* workflow, file conversion through the *Convert* workflow, and authenticated project management through the *Share* and project-management area. The chapter also identifies the relevant user roles, core user journeys, functional requirements, non-functional requirements, and deferred functionality belonging to the broader QDArchive vision.

3.1 Requirements Scope and Derivation

The QDArchive frontend requirements were derived through a staged process. The initial requirements were established before and during the early implementation phase based on the related work and the original QDArchive project description. During development, selected interaction-level details were refined through implementation experience, supervisor feedback, team discussions, and changes in the available backend functionality. These refinements were reflected in the updated QDArchive product requirements document (PRD) and clarified how previously identified system goals should be expressed through concrete frontend workflows; they were not derived retrospectively from whichever functionality happened to be completed.

The related work provided the conceptual foundation for the requirements. It showed that QDA artifacts differ from ordinary research files because they combine primary research data with analytical structures such as codes, coded segments, code systems, memos, annotations, and relationships. This creates frontend requirements for structured metadata, communication of artifact and visibility states, distinction between file roles, handling of missing referenced files, artifact inspection, and communication of format interoperability.

The original QDArchive project description defined the broader goal of a cloud-based service for archiving QDA artifacts, supporting review and verification, and enabling artifact interchange across different CAQDAS formats. The updated QDArchive PRD translated this broader vision into three principal user-facing solution areas: *Find*, *Convert*, and *Share*.[1] These solution areas provided the main structure for the frontend requirements and the information architecture of the prototype.

During implementation, the requirements were refined at the level of user interaction and workflow presentation. For example, public onboarding through the landing page, the authenticated dashboard, browser-local conversion history, and the three-step project-creation workflow were introduced or clarified in response to usability considerations and formative feedback. Such refinements extended the initial frontend specification while remaining consistent with the broader goals already established by the project documents and related work.

[1] The original QDArchive project description and the updated QDArchive PRD were used as internal project documents during requirements derivation. See Appendix A.

At the end of the implementation period, the refined requirements were consolidated into the final requirement baseline presented in this chapter. This baseline includes implemented, partially implemented, and deferred requirements. The implementation status of a requirement records how far the prototype realizes it; it does not determine whether the requirement belongs to the intended system. The evaluation in Chapter 6 compares the completed prototype against this final baseline rather than deriving the requirements from the evaluation results.

For this thesis, the scope is limited to the frontend prototype. Backend concerns such as long-term storage, conversion algorithms, access-control enforcement, backup management, administrator review queues, and background repository checks are considered only where they affect observable user-facing behavior. The frontend acts as the layer through which backend capabilities are presented and initiated through understandable workflows. Its scope includes public onboarding, authentication, archive search, conversion, project creation, authenticated project details, file management, project download, owner-facing sharing controls, versioning, and user settings.

The sources and refinement stages used to establish the final frontend requirements are summarized in Table 3.1. The internal project documents are not reproduced in full, but their role in the requirements process is documented in Appendix A.

The implemented prototype does not cover the complete long-term QDArchive vision. Reviewer-specific interfaces, administrator interfaces, complete publication workflows, advanced search and filtering, named-user sharing, Google authentication, data export, account deletion, notification preferences, and public artifact detail views remain deferred. They are nevertheless included in the final requirement baseline because they belong to the intended system and influence the extensibility expected from the frontend architecture.

3.2 User Roles

The QDArchive frontend addresses several user roles that differ in their access levels, goals, and interactions with QDA artifacts. The implemented prototype focuses mainly on visitors and registered researchers. Reviewers and administrators are included because they belong to the broader QDArchive vision, although dedicated interfaces for these roles remain future work. Table 3.2 summarizes the roles, their principal goals, their relevance to the frontend, and their prototype status.

Visitor / Anonymous User

A visitor is a user who is not authenticated. Visitors can access the landing page, browse and search public projects through the *Find* workflow, and use the *Convert* workflow without creating an account. This role is important because QDArchive should provide access to central value-providing workflows before requiring registration.

Researcher / Registered User

A researcher is an authenticated user who creates and manages QDA artifact projects. Researchers can access the dashboard, create projects, upload analysis files, add metadata and tags, upload missing referenced primary data files, manage project details, download project files, create project versions, and operate owner-facing sharing and visibility controls.

Source or refinement stage	**Role in requirements derivation**	**Frontend implication**
Related work	Established the domain-specific characteristics of QDA artifacts, including the combination of primary research data and analytical structures, interoperability limitations, and FAIR-oriented data-management needs.	Motivated metadata support, distinction between artifact components, missing-file handling, format communication, controlled access, and workflows supporting discovery and reuse.
Original QDArchive project description	Defined the broader system objective of archiving QDA artifacts, supporting review and verification, and enabling interchange across CAQDAS formats.	Motivated project creation, artifact upload, artifact inspection, project download, sharing, and format-conversion functionality.
Updated QDArchive PRD	Structured the user-facing platform around the three main solution areas *Find*, *Convert*, and *Share*.	Provided the main workflow and navigation structure for the frontend requirements and information architecture.
Iterative refinement during development	Clarified interaction-level requirements through implementation experience, supervisor and team feedback, usability considerations, and changes in available backend functionality.	Refined the landing page, dashboard, browser-local conversion history, project-detail presentation, and guided project-creation workflow.
Future QDArchive system vision	Preserved requirements belonging to the broader platform even when they were outside the completed prototype scope.	Identified deferred functionality such as reviewer and administrator interfaces, named-user sharing, complete publication workflows, notifications, data export, and account deletion.

Table 3.1: Sources and refinement stages used to establish the QDArchive frontend requirements

Reviewer

A reviewer is a user who is intended to receive controlled access to a QDA artifact, for example during a publication-review process. The current prototype provides owner-facing secure-link controls, but the recipient-facing secure-link view is incomplete. A dedicated reviewer interface therefore remains future work.

Administrator

An administrator is a privileged user responsible for operating and moderating the service. The broader QDArchive requirements include administrator-facing review and management functionality, but no administrator interface was implemented in this thesis prototype. This role is therefore treated as future work.

User role	**Main goal**	**Frontend relevance**	**Prototype status**
Visitor / Anonymous User	Find public artifacts and convert supported QDA files without first creating an account.	Landing page, public archive search, conversion workflow, legal pages, and sign-in or sign-up prompts.	Implemented
Researcher / Registered User	Create, manage, download, version, and share QDA artifact projects.	Authentication, dashboard, project creation, authenticated project details, file management, missing-file upload, download, versioning, owner-facing sharing controls, and user settings.	Partial
Reviewer	Inspect shared QDA artifacts under controlled access conditions.	Recipient-facing secure access, artifact inspection, and future reviewer-oriented workflows.	Future work
Administrator	Operate and moderate the service, including project and user management tasks.	Future administrator interfaces for review queues, project-status management, user management, and service moderation.	Future work

Table 3.2: User roles, goals, frontend relevance, and prototype status in QDArchive

3.3 Core User Journeys

The core user journeys describe how the roles identified in Section 3.2 interact with the QDArchive frontend. They are derived from the three principal user-facing solution areas of the updated PRD, namely *Find*, *Convert*, and *Share*, and from interaction-level refinements introduced during frontend development. In the prototype, *Share* is understood broadly as the authenticated project-management area, including project creation, file management, versioning, download, and owner-facing sharing controls.

The journeys are divided into public workflows, authenticated researcher workflows, and future workflows. Public workflows are available without registration and include landing-page access, archive search, and file conversion. Authenticated workflows include dash-

board access, project creation, project management, missing-file completion, versioning, download, and owner-facing sharing controls. Reviewer and administrator workflows belong to the broader QDArchive vision but were not implemented as complete recipient-facing or role-specific interfaces. Figure 3.1 illustrates this structure, while Table 3.3 summarizes the main journeys, roles, expected outcomes, and prototype status.

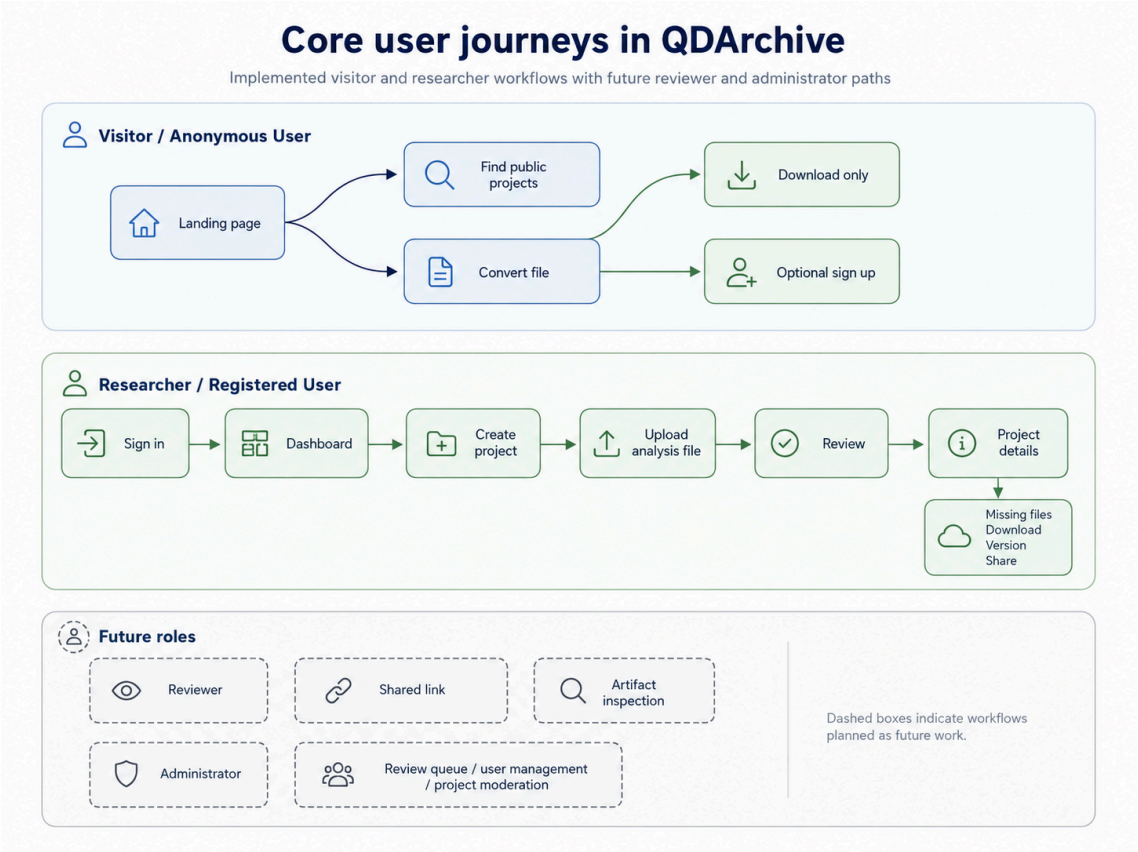


Figure 3.1: Core frontend user journeys in QDArchive

Landing and onboarding. The landing page is the public entry point into QDArchive. It introduces the purpose of the platform, presents the main features, explains the basic idea of QDA artifacts, and provides links to the principal workflows. Although the landing page was not part of the original core technical requirements, it was added as a frontend design decision to improve discoverability, onboarding, trust, and future adoption. From this page, visitors can access *Find* and *Convert* or continue to registration and login.

Finding public artifacts. The *Find* journey allows visitors and authenticated users to search and browse projects available through the archive. Users can search by name, description, or keywords, switch between available project scopes, and sort the result list. If an unauthenticated user selects the *My projects* filter, the frontend prompts the user to sign in. The current prototype supports project listing and basic archive exploration, while advanced filtering and a working public artifact detail view remain future work.

Converting artifacts. The *Convert* journey allows users to convert supported artifact files through the integrated conversion service. Users upload a file through drag-and-drop or manual selection, review the detected format, manually select the format if automatic

detection fails, choose a target format, accept the Terms of Service, start the conversion, and download the result. Unauthenticated users can complete this workflow without creating an account. They are also offered the option to register after conversion. Recent conversions remain available in the same browser profile until the local history is cleared or replaced.

Creating and managing projects. Project creation is available to authenticated users. It follows a guided three-step workflow consisting of metadata entry, analysis-file upload, and final review. The user provides a project name, optional description, and tags, uploads the main analysis file, confirms or corrects the detected format where necessary, and reviews the information before completing project creation. New projects are private by default. After creation, the user is redirected to the authenticated project detail page.

Completing and inspecting project files. After a project has been created, QDArchive identifies the analysis file and any primary data files referenced by it. If referenced files are missing, the project detail page displays a warning and provides upload actions. The frontend distinguishes between the analysis file, referenced primary data files, and generic supplementary files. Metadata and file-management operations are available, although direct viewing of uploaded files is defective in the current prototype.

Downloading and versioning projects. Authenticated users can download project material and create new project versions. The download workflow allows users to select an export format, choose which files should be included, and select the archive format where supported. The versioning workflow allows an existing project to be updated while retaining or replacing selected files.

Visibility, sharing, and publication. Authenticated project owners can change a project between private and public visibility and can generate and manage secure share links. These controls represent owner-facing access configuration. They do not yet constitute a complete publication or sharing workflow because recipient-facing public and secure-link pages are incomplete.

In this thesis, setting a project to *public* refers only to changing its visibility state and generating the corresponding owner-facing public URL or archive visibility. It does not mean that the project has completed a full publication process. Complete publication would additionally require a working public artifact detail page, open-licence selection, any required metadata validation, and possible review or approval steps.

Future reviewer and administrator journeys. Reviewer and administrator journeys belong to the broader QDArchive vision but were not implemented as dedicated interfaces. Reviewers are expected to inspect shared artifacts through working controlled-access views, for example during publication review. Administrators are expected to manage review queues, project states, users, and moderation operations. These journeys are therefore included as deferred requirements.

3.4 Functional Requirements

The functional requirements describe the user-facing behavior expected from the QDArchive frontend. They are organized according to the main workflow areas: public entry and au-

Journey	**Main user role**	**Main steps**	**Expected outcome**	**Prototype status**
Landing and onboarding	Visitor	Open the landing page, read feature sections, access legal pages, and choose *Find*, *Convert*, sign in, or sign up.	The user understands the purpose of QDArchive and selects an appropriate entry point.	Implemented
Find artifact	Visitor / Researcher	Open *Find*, enter a search query, switch between available project scopes, sort results, and inspect the result list.	The user discovers public projects or is prompted to sign in for authenticated project access.	Partial
Convert artifact	Visitor / Researcher	Upload an artifact file, confirm or select its format, choose a target format, accept the Terms of Service, convert the file, and download the result.	The user receives a converted artifact and can download it.	Implemented
Create project	Researcher	Open the dashboard, start project creation, enter metadata, add tags, upload an analysis file, review the information, and complete creation.	A private QDArchive project is created and associated with the authenticated user.	Implemented
Manage project	Researcher	Open the authenticated project detail page and inspect or edit project metadata, file roles, missing-file states, and supplementary files.	The user can understand and manage the available structure of the QDA artifact.	Partial
Complete missing files	Researcher	Review missing referenced files, upload the required primary data files, and edit supported file information.	The project contains the referenced primary files required by the represented artifact structure.	Implemented
Download and version	Researcher	Choose download options, select included files, create a new project version, and retain or replace existing files.	The user can retrieve and update the project material.	Implemented
Configure visibility and sharing	Researcher	Generate or manage a secure share link and change a project between private and public visibility.	The owner can configure access settings, although recipient-facing access remains incomplete.	Partial
Review shared artifact	Reviewer	Open a shared link and inspect artifact metadata, files, and analytical material.	The reviewer can inspect shared research material under controlled conditions.	Future work
Admin review and management	Administrator	Review submitted or externally discovered projects, manage project states and users, and perform moderation tasks.	The administrator can operate and moderate the QDArchive service.	Future work

Table 3.3: Core user journeys of the QDArchive frontend

thentication, archive search, conversion, authenticated project management, user settings, and deferred reviewer and administrator functionality.

Each requirement is assigned an identifier, a priority, and a prototype implementation status. These two classifications describe different properties:

- **Priority** indicates the importance of the requirement to the intended QDArchive workflows. A high-priority requirement is central to the core workflow, a medium-priority requirement provides important supporting functionality, and a low-priority requirement is a desirable extension that does not prevent the central workflows from operating.
- **Prototype status** indicates how far the requirement was realized in the frontend prototype. *Implemented* means that the required frontend behavior is available, *Partial* means that only part of the required behavior is available or that a relevant defect remains, and *Future work* means that the requirement belongs to the intended system but was not completed in the prototype.

Priority and prototype status are independent. A high-priority requirement may remain partial when technical dependencies or implementation constraints prevent its completion, while a lower-priority requirement may already be implemented.

3.4.1 Public Entry, Authentication, and Legal Pages

The public entry and authentication requirements define how users access QDArchive before and during account creation. They cover the landing page, public navigation, legal pages, registration, login, logout, and password recovery. Table 3.4 summarizes these requirements.

3.4.2 Find / Archive Search

The *Find* requirements define how users search and browse QDArchive projects. The workflow is available to visitors and authenticated users. The current prototype supports basic search, scope selection, sorting, result display, authentication prompts, and empty states. Advanced filtering and complete public artifact inspection remain outside the completed prototype scope. Table 3.5 summarizes the requirements for this workflow.

ID	Requirement	Priority	Prototype status
FR-PE1	The frontend shall provide a public landing page introducing QDArchive and its main workflows.	High	Implemented
FR-PE2	The frontend shall provide clear navigation entry points to *Find*, *Convert*, *Share*, sign in, and sign up.	High	Implemented
FR-PE3	The frontend shall provide legal pages for the Terms of Service, Privacy Policy, Cookie Policy, Contact / Imprint, and Third-Party Notices.	Medium	Implemented
FR-PE4	The frontend shall allow users to create an account using basic profile information and email/-password credentials.	High	Implemented
FR-PE5	The frontend shall require users to accept the Terms of Service and Privacy Policy during registration.	High	Implemented
FR-PE6	The frontend shall allow users to sign in, sign out, request a password reset, and set a new password through the supported recovery flow.	High	Implemented
FR-PE7	The frontend should support external identity providers such as Google sign-in in the future.	Low	Future work

Table 3.4: Functional requirements for public entry, authentication, and legal pages

ID	Requirement	Priority	Prototype status
FR-F1	The frontend shall provide a *Find* page for searching and browsing public QDA projects.	High	Implemented
FR-F2	The *Find* page shall be accessible without authentication.	High	Implemented
FR-F3	The frontend shall allow users to search projects by name, description, or keywords.	High	Implemented
FR-F4	The frontend shall allow users to switch between all visible projects and their own projects where applicable.	Medium	Implemented
FR-F5	If an unauthenticated user selects the *My projects* filter, the frontend shall prompt the user to sign in.	High	Implemented
FR-F6	The frontend shall provide basic sorting options such as newest first, oldest first, recently updated, and name A–Z.	Medium	Implemented
FR-F7	The frontend shall display search results with relevant project information such as title, description, tags, source, and date where available.	High	Implemented
FR-F8	The frontend shall provide clear empty states when no projects match the current search or filter settings.	High	Implemented
FR-F9	The frontend should allow users to open a detailed public artifact view directly from a *Find* result.	Medium	Future work
FR-F10	The frontend should provide advanced filters for artifact type, licence, file format, and other metadata.	Medium	Future work

Table 3.5: Functional requirements for the Find / archive-search workflow

3.4.3 Convert

The *Convert* requirements define how users submit artifact files to the integrated conversion service and retrieve converted results. This workflow is intentionally available to authenticated and unauthenticated users in order to reduce entry barriers. It includes upload, format detection, manual format selection, target-format selection, acceptance of the Terms of Service, conversion feedback, download, and browser-local conversion history.

Throughout this thesis, *QDA* without a preceding dot refers to qualitative data analysis or to QDA artifacts as a general concept. The notation `.qda` refers specifically to a file extension exposed by the integrated `qda-convert` library.

The prototype exposes the following relevant format labels and extensions:

- `.qdpx` denotes the REFI-QDA project exchange package defined by the REFI-QDA standard.
- `.refi` is a label and extension exposed by the integrated conversion library for a REFI-related representation. It does not denote a separate interoperability standard from REFI-QDA. References to *REFI* and `.qdpx` in the implementation should therefore not be interpreted as two unrelated standards.
- `.qda` denotes a library-specific QDA corpus representation provided by `qda-convert`. The internal corpus model, including the library's corpus or CRPS representation, and the correctness of the conversion algorithms were developed outside the scope of this frontend thesis.

The frontend contribution is limited to accepting the format labels supplied by the conversion service, communicating them to users, collecting source and target selections, invoking the conversion operation, and presenting the result. It does not define the underlying formats or conversion semantics. Table 3.6 summarizes the functional requirements for this workflow.

Table 3.6: Functional requirements for the Convert workflow

ID	Requirement	Priority	Prototype status
FR-C1	The frontend shall provide a conversion page for converting artifact files between formats supported by the integrated conversion service.	High	Implemented
FR-C2	The conversion workflow shall be available to authenticated and unauthenticated users.	High	Implemented
FR-C3	The frontend shall support drag-and-drop upload and manual file selection.	High	Implemented
FR-C4	The frontend shall display the selected file name and the detected format label or extension after upload.	High	Implemented

Continued on next page

Table 3.6 – continued from previous page

ID	Requirement	Priority	Prototype status
FR-C5	If the file format cannot be detected automatically, the frontend shall prompt the user to select the appropriate source format manually.	High	Implemented
FR-C6	The frontend shall allow users to remove the selected file before starting conversion.	Medium	Implemented
FR-C7	The frontend shall allow users to choose a supported target format and shall prevent the detected source format from being selected as the target.	High	Implemented
FR-C8	The frontend shall prevent conversion until a valid file, target format, and required acceptance of the Terms of Service are provided.	High	Implemented
FR-C9	The frontend shall require users to accept the Terms of Service before conversion because uploaded files are processed through the QDArchive service and may be represented as private QDArchive projects.	High	Implemented
FR-C10	The frontend shall display a success state and provide a download action after successful conversion.	High	Implemented
FR-C11	The frontend shall provide a recent conversions view for downloading converted files again in the same browser profile until the local history is cleared or replaced.	Medium	Implemented
FR-C12	The frontend shall allow unauthenticated users to download a converted file without creating an account.	High	Implemented
FR-C13	The frontend shall offer unauthenticated users the option to create an account after conversion in order to save and manage the converted artifact later.	Medium	Implemented
FR-C14	For authenticated users, converted artifacts shall be associated with the signed-in account as private projects where supported by the backend.	Medium	Implemented

3.4.4 Authenticated Project Management and Share

The authenticated project-management requirements define the principal researcher workflow of the prototype. In this thesis, the *Share* area is understood broadly as the authenticated area in which users create, manage, prepare, download, version, and configure access

to QDA artifact projects. This includes the dashboard, guided project creation, authenticated project detail pages, file organization, missing-file handling, versioning, download, visibility changes, and secure-link controls.

A distinction is made between *public visibility* and *complete publication*. Public visibility indicates that the owner has changed the project's access state and that the project may be included in public discovery or assigned a public URL. This state does not by itself fulfil the complete publication requirement. Complete publication additionally requires a working public artifact view, open-licence selection, any required metadata validation, and possible review or approval processes.

Similarly, secure-link management from the owner interface does not by itself constitute successful controlled access. The complete requirement also depends on the recipient being able to open and inspect the project through the generated link. Table 3.7 summarizes the requirements for authenticated project management and sharing.

Table 3.7: Functional requirements for authenticated project management and Share

ID	**Requirement**	**Priority**	**Prototype status**
FR-PM1	The frontend shall provide an authenticated dashboard for registered users.	High	Implemented
FR-PM2	The dashboard shall provide quick actions for finding public projects, creating a project, and browsing the user's projects.	High	Implemented
FR-PM3	The dashboard shall display the user's recent projects and provide access to the full project list.	Medium	Implemented
FR-PM4	The dashboard should display starred or favourite projects.	Low	Future work
FR-PM5	The frontend shall provide a guided workflow for creating and uploading a QDArchive project.	High	Implemented
FR-PM6	The project-creation workflow shall collect a required project name and an optional description.	High	Implemented
FR-PM7	The frontend shall allow users to add up to ten tags to a project to support search and filtering.	Medium	Implemented
FR-PM8	The frontend shall allow users to upload a CAQDAS project or exchange file as the main analysis artifact.	High	Implemented
FR-PM9	The frontend shall detect the uploaded analysis-file format where possible and prompt for manual format selection if detection fails.	High	Implemented

Continued on next page

Table 3.7 – continued from previous page

ID	Requirement	Priority	Prototype status
FR-PM10	The frontend shall provide a review step before project creation is finalized.	High	Implemented
FR-PM11	Newly created projects shall be private by default.	High	Implemented
FR-PM12	After successful project creation, the frontend shall redirect the user to the created project's authenticated detail page.	High	Implemented
FR-PM13	The authenticated project detail page shall display project metadata including title, version, creation date, access type, source, uploader, description, and tags.	High	Implemented
FR-PM14	The frontend shall distinguish between analysis files, referenced primary data files, and generic supplementary files.	High	Implemented
FR-PM15	The frontend shall identify primary data files referenced by the uploaded analysis artifact where this information is provided by the backend.	High	Implemented
FR-PM16	The frontend shall clearly indicate missing referenced files and guide users to upload them.	High	Implemented
FR-PM17	The frontend shall allow users to upload missing referenced primary data files.	High	Implemented
FR-PM18	The frontend shall allow users to edit supported project metadata and file information.	Medium	Implemented
FR-PM19	The frontend shall provide a file-centred overview for inspecting and managing project files.	Medium	Partial
FR-PM20	The frontend shall allow users to create new versions of an existing project.	Medium	Implemented
FR-PM21	The frontend shall allow users to download project artifacts.	High	Implemented
FR-PM22	The download workflow shall allow users to choose an export format, select included files, and choose the archive format where supported.	High	Implemented
FR-PM23	The frontend shall support controlled link-based access by allowing project owners to generate, copy, disable, enable, and regenerate secure share links and by providing a working recipient-facing secure view.	High	Partial

Continued on next page

Table 3.7 – continued from previous page

ID	Requirement	Priority	Prototype status
FR-PM24	The frontend shall allow project owners to change a project between private and public visibility where supported. This visibility change shall not be presented as completion of the full publication workflow.	Medium	Implemented
FR-PM25	The frontend should provide a working public artifact detail view for public project links.	Medium	Future work

3.4.5 User Settings

The user-settings requirements define how authenticated users manage account-related information. The implemented prototype supports basic profile editing and profile-image management. Notification settings, account-data export, and account deletion are represented as planned functionality but are not supported by complete frontend and backend operations. Table 3.8 summarizes these requirements.

ID	Requirement	Priority	Prototype status
FR-US1	The frontend shall provide a settings area for authenticated users.	Medium	Implemented
FR-US2	The frontend shall allow users to view their profile information.	Medium	Implemented
FR-US3	The frontend shall allow users to update optional profile fields such as first name, last name, country, and city or state.	Medium	Implemented
FR-US4	The frontend shall display the registered email address as account information while preventing direct editing of the email field.	Medium	Implemented
FR-US5	The frontend shall allow users to upload, change, and remove a profile image.	Low	Implemented
FR-US6	The frontend should provide notification preferences in the future.	Low	Future work
FR-US7	The frontend should provide account-data export once the required backend support exists.	Low	Future work
FR-US8	The frontend should provide account deletion once the required backend support exists.	Low	Future work

Table 3.8: Functional requirements for user settings

3.4.6 Deferred Reviewer, Administrator, and Publication Requirements

Reviewer, administrator, and complete publication functionality belongs to the broader QDArchive system vision but was not fully implemented as part of this thesis prototype.

These requirements remain relevant because they influence how the frontend should evolve and which extension points should be preserved.

In particular, a complete publication workflow is broader than changing a project's visibility to public. It requires a working recipient-facing public artifact view, open-licence selection, appropriate metadata and access information, and any review or approval stages required by the final QDArchive service. Table 3.9 lists the deferred requirements.

ID	Requirement	Priority	Prototype status
FR-DA1	The frontend should provide a reviewer-oriented interface for inspecting shared QDA artifacts.	Medium	Future work
FR-DA2	The frontend should support named-user sharing by allowing researchers to invite specific users by email.	Medium	Future work
FR-DA3	The frontend should support a complete publication workflow, including open-licence selection, metadata review, publication into the public archive, and a working public artifact detail view.	Medium	Future work
FR-DA4	The frontend should provide an administrator interface for reviewing submitted or externally discovered projects.	Low	Future work
FR-DA5	The frontend should provide administrator tools for managing users, project status, and moderation tasks where required.	Low	Future work

Table 3.9: Deferred functional requirements for reviewer, administrator, and publication workflows

3.5 Non-Functional Requirements

Non-functional requirements describe the quality attributes that the QDArchive frontend should satisfy across its workflows. Unlike functional requirements, they are not limited to a single feature or page. They define how the frontend should behave in terms of usability, transparency, accessibility, responsiveness, privacy awareness, maintainability, and extensibility.

These requirements are particularly important for QDArchive because users interact with artifacts that may contain analysis files, referenced primary data, supplementary files, metadata, missing references, versions, and multiple visibility states. The non-functional requirements in Table 3.10 provide the quality criteria used to guide the frontend design and evaluate the implemented prototype.

Table 3.10: Non-functional requirements for the QDArchive frontend

ID	Quality attribute	Requirement	Priority
NFR-1	Usability	The frontend shall make the principal QDArchive workflows understandable to users who may not be familiar with CAQDAS file formats or repository systems.	High
NFR-2	Learnability	The frontend shall guide users through complex workflows using clear labels, helper text, step indicators, empty states, review screens, and contextual explanations.	High
NFR-3	Transparency	The frontend shall clearly communicate relevant project and file states, including file format, access type, missing files, project visibility, conversion status, and available actions.	High
NFR-4	Privacy awareness	The frontend shall clearly distinguish between private, securely shared, and public project states so that users understand the intended access conditions.	High
NFR-5	Low entry barrier	The frontend shall allow visitors to access central value-providing workflows, particularly archive search and file conversion, before requiring account creation.	High
NFR-6	Responsiveness	The frontend shall be usable across common desktop, tablet, and mobile viewport sizes, with layouts adapting to the available screen width.	High
NFR-7	Accessibility	The frontend should use readable typography, sufficient contrast, semantic structure, keyboard-accessible controls, and understandable form labels where possible.	Medium
NFR-8	Feedback and error handling	The frontend shall provide clear feedback for loading states, successful actions, errors, empty states, disabled actions, and missing required input.	High
NFR-9	Consistency	The frontend shall use consistent terminology, navigation patterns, layouts, buttons, cards, icons, and status indicators across workflows.	Medium
NFR-10	Interoperability communication	The frontend shall communicate supported file extensions and library labels, including `.qdpx`, `.refi`, and `.qda`, as well as detected source formats, available target formats, conversion limitations, and missing referenced files in a way that is understandable to users.	High

Continued on next page

Table 3.10 – continued from previous page

ID	Quality attribute	Requirement	Priority
NFR-11	Maintainability	The frontend shall be structured using reusable components, typed data models, and separated service logic to support future development and reduce duplication.	High
NFR-12	Extensibility	The frontend shall be designed so that future workflows, such as reviewer interfaces, administrator interfaces, notification settings, data export, account deletion, and complete publication workflows, can be added without changing the overall interaction model.	Medium

Together, these non-functional requirements emphasize that the frontend should not only expose the available system functionality but also make the complexity of QDA artifact management understandable. Transparency, privacy awareness, and interoperability communication are particularly important because users need to understand which type of file they are uploading, how format labels relate to one another, which referenced files are missing, how project access is configured, and which conversion or download options are available.

3.6 Summary

This chapter established the final requirement baseline for the QDArchive frontend prototype. The requirements were initially derived from the related work and original QDArchive project description and were refined through the updated PRD, implementation experience, supervisor and team feedback, usability considerations, and changes in available backend functionality. The resulting baseline was consolidated independently of whether each requirement was ultimately completed.

The relevant roles are visitors, registered researchers, reviewers, and administrators. The prototype focuses mainly on visitors and registered researchers. Visitors can use the landing page, archive search, and conversion workflow without prior registration. Registered researchers can use the dashboard, create and manage projects, upload missing referenced files, edit metadata, create versions, configure downloads, and operate owner-facing visibility and secure-link controls. Reviewer and administrator interfaces remain future work.

The functional requirements were grouped into public entry and authentication, *Find*, *Convert*, authenticated project management and sharing, user settings, and deferred reviewer, administrator, and publication workflows. Priority was distinguished from prototype status: priority records the importance of a requirement to the intended system, whereas prototype status records how far it was realized in the implementation.

The chapter also clarified the format terminology used by the frontend. The extension `.qdpx` denotes the REFI-QDA project exchange package, while `.refi` is a REFI-related label exposed by the integrated conversion library rather than a separate standard. The extension `.qda` denotes a library-specific representation provided by `qda-convert`. The

internal corpus representation and conversion algorithms fall outside the scope of this frontend thesis.

A further distinction was made between public visibility and complete publication. Changing a project to public is an owner-facing visibility operation. It does not satisfy the complete publication requirement without a working public artifact detail view, open-licence selection, metadata validation, and any necessary review or approval stages.

The non-functional requirements emphasize usability, learnability, transparency, privacy awareness, responsiveness, accessibility, maintainability, extensibility, feedback, consistency, and clear communication of interoperability-related information. The following chapter uses this requirement baseline to explain the frontend design, including the information architecture, interaction flows, responsive strategy, visual structure, and principal user-interface decisions.

4 System Design of the QDArchive Frontend

Chapter 3 established the requirements that the QDArchive frontend prototype should support. This chapter explains how those requirements were translated into the system design of the user-facing application. The discussion covers the design goals and constraints, application architecture, technology choices, information architecture, user experience design, responsive behavior, and visual identity.

The QDArchive application contains both browser-executed user-interface code and server-executed functionality within the Next.js application. This chapter focuses primarily on the frontend and its interaction with those server-side capabilities. System-level responsibilities such as persistent storage, authorization enforcement, conversion execution, archive generation, and long-term data management are distinguished from the routes, pages, components, forms, and interaction patterns through which users access them.

The purpose of the frontend design is therefore not merely to expose technical operations. It must translate the domain-specific complexity of QDA artifacts into understandable workflows for visitors and registered researchers, while providing an extensible structure for future reviewer and administrator functionality.

Representative screenshots are included to illustrate selected design decisions. A broader gallery of the implemented prototype is provided in Appendix B.

4.1 Design Goals and Constraints

The frontend design was guided by the functional and non-functional requirements defined in Chapter 3. The central design goal was to provide a clear interface for finding, converting, creating, managing, downloading, versioning, and sharing QDA artifacts. Because these artifacts may combine analysis files, referenced primary data, supplementary files, metadata, versions, missing references, and different visibility states, the frontend needed to communicate artifact structure and project state without requiring users to understand the underlying storage or conversion implementation.

A second goal was to reduce the initial entry barrier. The *Find* and *Convert* workflows were designed as public workflows that do not require prior registration. Visitors can therefore explore the archive and use file conversion before deciding whether to create an account. Authenticated functionality is introduced when users need to create persistent projects, manage files and metadata, create versions, configure downloads, or operate visibility and sharing controls.

A third goal was to support progressive complexity. Simple entry points are provided through the landing page, public navigation, and authenticated dashboard. More complex functionality is introduced only within the workflow where it becomes relevant. For example, project creation separates metadata entry, analysis-file upload, and final review into three steps. The project detail area further separates project metadata, analysis files, referenced primary files, supplementary files, missing-file handling, version management, download configuration, and access controls.

The design was constrained by the evolving state of the broader QDArchive system. Conversion, authentication, project creation, project management, versioning, downloads, file operations, and owner-facing visibility and secure-link controls could be connected to available services. Other functions, including recipient-facing public and secure-link

views, domain-specific artifact inspection, reviewer and administrator interfaces, named-user sharing, account-data export, account deletion, notifications, and a complete licence-aware publication workflow, remained incomplete or unavailable.

Planned functionality must be visually distinguishable from operational functionality. A control that resembles an active button or completed feature can otherwise mislead users and distort the evaluation of the prototype. Where future functionality remains visible in the interface, it should therefore be disabled and accompanied by an explicit message such as *Not available yet*. A disabled visual style alone is not considered sufficient communication.

The relationship between broader system responsibilities and frontend design responsibilities is summarized in Table 4.1.

Design aspect	**System-level responsibility**	**Frontend design responsibility**
Main concern	Persistence, authorization enforcement, data management, service boundaries, conversion execution, and archive generation.	User workflows, navigation, page structure, interaction design, validation, and visual communication.
Find, *Convert*, and *Share*	Backend and application capabilities supporting search, conversion, project management, download, and sharing.	User-facing workflows that guide visitors and researchers through the available operations.
Artifact handling	Structured representation of projects, files, references, versions, metadata, and lifecycle states.	Authenticated project detail pages, file sections, missing-file warnings, metadata editing, upload controls, and download configuration.
Access and visibility	Authentication, authorization rules, access tokens, and enforcement of project access conditions.	Communication of private, public, and securely shared states and presentation of owner-facing access controls.
Conversion	Parsing, conversion algorithms, validation, and generation of converted output.	Source-file upload, format communication, target-format selection, conversion initiation, progress feedback, and result download.
Quality focus	Reliability, security, data integrity, persistence, and technical interoperability.	Usability, learnability, transparency, responsiveness, accessibility, consistency, and extensibility.

Table 4.1: Separation between system-level and frontend design responsibilities in QDArchive

The main constraints can therefore be summarized as follows. First, the frontend had to follow the QDArchive product structure around *Find*, *Convert*, and *Share*. Second, it had to remain compatible with services and backend capabilities that evolved during development. Third, it had to distinguish operational functionality from unavailable or planned functionality. Fourth, it had to provide consistent interaction patterns for complex QDA artifact structures while remaining understandable to users with different levels of technical and CAQDAS experience.

4.2 Application Architecture and Technology Choices

The QDArchive application was implemented using Next.js, React, and TypeScript. Next.js provides the application framework, routing structure, layouts, server-side routes, and runtime environment. React provides the component-based user-interface layer, while TypeScript provides static typing for models, service inputs and outputs, component properties, and utilities.

The application uses the Next.js App Router and route groups such as `(auth)`, `(marketing)`, and `(protected)`. Route groups organize pages by access level and layout responsibility without adding the group name to the public URL (Vercel, 2026).

The codebase separates routes, feature components, query hooks, query keys, service functions, contexts, models, utilities, and theme configuration. The `app` directory contains the application route structure. Public and marketing-related pages are grouped under `(marketing)`, authentication pages under `(auth)`, and authenticated pages under `(protected)`.

The dashboard is organized under:

- `(protected)/(dashboard)/dashboard`

Authenticated project and account functionality is organized under:

- `(protected)/(app)/projects/[id]`
- `(protected)/(app)/projects/upload`
- `(protected)/(app)/settings`

Public archive and conversion workflows are available through the `archive` and `convert` routes. The route structure shown in Figures 4.1 and 4.2 was verified against the final application route tree. The presence of a route does not necessarily imply that its complete workflow is operational; for example, recipient-facing public and secure project routes exist in the route structure, but their final views remain incomplete.

The component structure follows the functional areas of the application. Components are grouped into feature-oriented modules such as `archive`, `auth`, `convert`, `dashboard`, `marketing`, `projects`, and `settings`. Shared layout components and common interface elements are separated from feature-specific components. This organization keeps the codebase aligned with the user-facing workflows while supporting component reuse.

Communication with server-side functionality is organized through service modules and TanStack React Query hooks. Service modules such as `ArchiveService`, `AuthService`, `ConvertService`, `FileService`, `ProjectService`, `ProjectVersionService`, `ShareLinkService`, and `UserService` encapsulate requests to Next.js API routes and available backend services.

React Query hooks call these service methods to retrieve and mutate server data. Query keys are defined separately and are used for caching, refetching, invalidation, and sharing query results across components (TanStack, 2026). This structure reduces direct API logic inside pages and supports consistent cache updates after project, file, version, profile, and sharing operations.

The application also uses context providers for cross-cutting concerns. `AuthContext` exposes authentication-related state, `SupabaseContext` provides access to the configured

Supabase client, and `ToastContext` supports user-facing success and error notifications. These contexts keep global concerns separate from individual feature components.

The architecture is illustrated in Figure 4.1. The diagram presents the main relationship between routes, feature components, query hooks, services, server-side routes, and backend services.

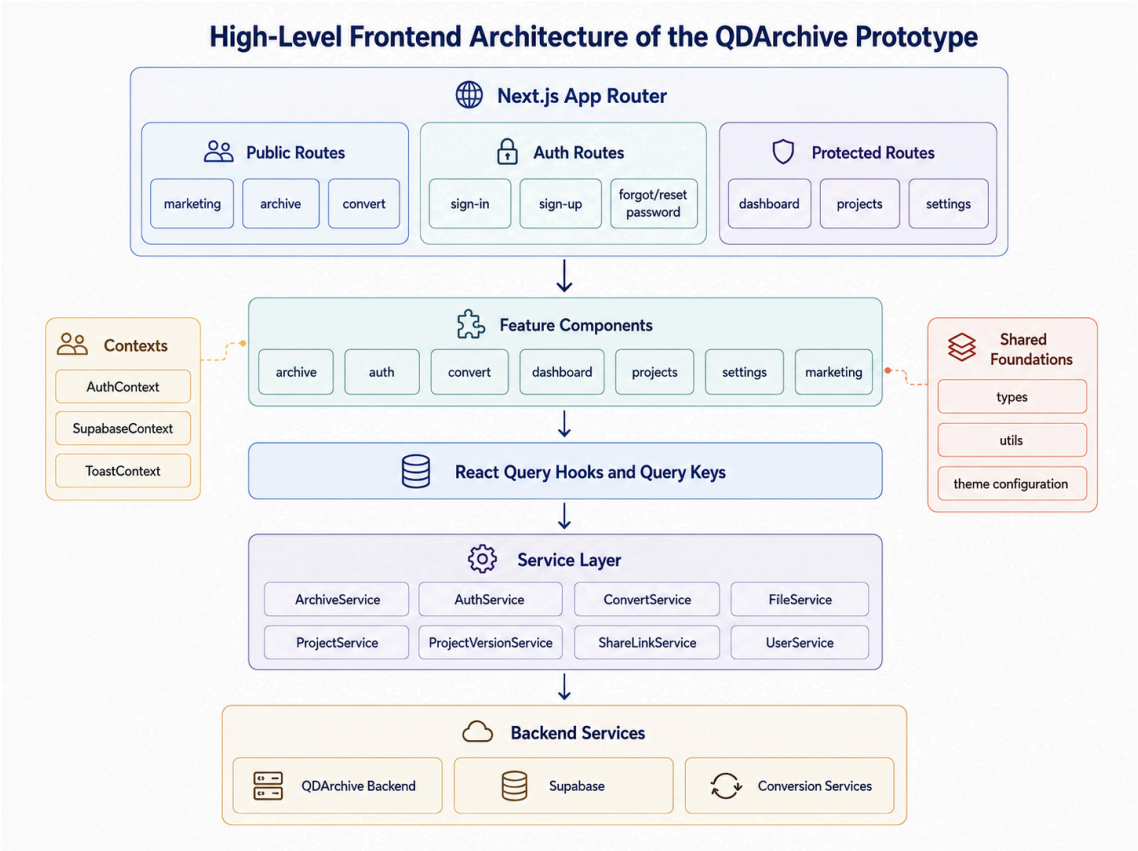


Figure 4.1: High-level application architecture of the QDArchive prototype

4.2.1 Material UI and Tailwind CSS Responsibilities

Material UI was selected as the primary component library because it provides a mature and broad collection of React components for forms, dialogs, navigation, cards, tables, feedback states, and responsive layouts. Its Material Design foundations were compatible with the intended QDArchive identity and provided a stable basis for implementing consistent interaction behavior across the prototype.

The component library also provides a centralized theme system for palette values, typography, spacing, shape, breakpoints, and component-level overrides (MUI, 2026). This made it possible to align standard components with the QDArchive visual identity without implementing basic controls independently.

The shadcn/ui approach was considered briefly during technology selection. It was not adopted because the prototype required a comparatively broad ready-made component base and consistent support for forms, dialogs, data presentation, navigation, and feedback. Material UI provided this broader foundation with less need to introduce or maintain additional component implementations.

Tailwind CSS is used alongside Material UI, but the two technologies have different responsibilities:

- **Material UI** provides the principal interactive components, their behavior, accessibility-related defaults, component states, and the authoritative visual theme.
- **Emotion** provides the styling engine used by Material UI and supports theme-aware component customization.
- **Tailwind CSS** provides lightweight utility classes for general layout, spacing, sizing, alignment, responsive adjustments, and styling that does not require a dedicated Material UI component.

Tailwind therefore replaces repeated general-purpose CSS declarations rather than replacing the Material UI component system. Its use also preserves a utility-based styling layer that can support later theme and layout extensions. However, the Material UI theme remains the primary source of component-level colors, typography, shapes, and interactive states. Maintaining this separation reduces the risk of conflicting styling responsibilities.

4.2.2 Browser-Side and Server-Side Dependencies

Not every dependency used by the QDArchive application executes in the browser. Although the application is described as a frontend prototype, the Next.js codebase includes server-side routes that support user-facing workflows.

The QDConvert package is installed as a dependency through the application's package configuration but executes on the server side of the Next.js application. Its source code is maintained separately and does not reside in the QDArchive frontend and backend repository. The frontend invokes the conversion workflow through the application's service and API layers; it does not execute the conversion library directly in the browser.

Similarly, Archiver is used by server-side application code to generate downloadable archives such as ZIP and TAR.GZ packages. It is not shipped as part of the browser interface. Describing these packages simply as frontend libraries would therefore be misleading. They are application dependencies that support frontend-initiated workflows from the server execution context.

The principal technologies and their execution contexts are summarized in Table 4.2. The versions were verified against the final project dependency configuration and lockfile.

The selected technologies support the design goals of maintainability, consistency, and extensibility. Next.js and React provide the page, route, and component model. TypeScript supports consistent data contracts across user-interface and service code. React Query separates remote server state from temporary component state. The service layer centralizes API communication, while Material UI, Emotion, Tailwind CSS, and shared design tokens provide a structured styling system.

Temporary interface state, including form values, selected files, open dialogs, and active workflow steps, is managed locally within the relevant components or workflow contexts. Remote project, file, version, archive, profile, and sharing data are managed through React Query.

Recent guest conversions are stored through browser-local storage rather than session-only state. They can therefore remain available in the same browser profile beyond a single browser session until the stored history is cleared or replaced by newer entries.

Technology	Version	Execution context	Role in the application
Next.js	16.0.7	Browser and server	Provides the application framework, routing, layouts, server-side routes, rendering, and runtime environment.
React	19.2.4	Browser	Provides the component model used to construct pages, forms, cards, dialogs, workflow steps, and reusable interface elements.
TypeScript	5.x	Build-time / application-wide	Provides static typing for models, components, services, API inputs and outputs, and utilities.
Material UI	7.3.5	Browser	Provides the primary component library and theme system for forms, buttons, dialogs, cards, navigation, tables, and feedback states.
Emotion	11.14.x	Browser	Provides the styling engine used by Material UI and supports component-level theme integration.
Tailwind CSS	4.x	Build-time and browser styling	Provides utility classes for layout, spacing, sizing, alignment, responsive adjustments, and general non-component styling.
TanStack React Query	5.90.20	Browser	Manages server state, data fetching, caching, mutations, invalidation, and synchronization for archive, conversion, project, file, version, profile, and sharing workflows.
Supabase libraries	2.81.1 / 0.7.0	Browser and server	Support authentication and configured client-side and server-side Supabase integration.
QDConvert package	2.0.1	Server	Executes conversion-related operations through server-side Next.js routes. The library is maintained outside the QDArchive repository.
Archiver	7.0.1	Server	Generates downloadable archive packages, including ZIP and TAR.GZ output, for frontend-initiated download workflows.
Phosphor Icons	2.1.10	Browser	Provides consistent icons for navigation, actions, upload areas, files, settings, status indicators, and warnings.
Source Serif 4 and Source Sans 3	5.2.9	Browser	Provide the heading and body typography used by the QDArchive visual identity.

Table 4.2: Main application technologies and their execution contexts

The architecture also supports future extension. Reviewer and administrator workflows can be introduced through additional protected routes and feature modules. Notification settings, data export, account deletion, dark mode, working public artifact views, and further inspection functionality can build on the existing route, component, service, query, and theme structure.

4.3 Information Architecture

The information architecture translates the workflow areas identified in Chapter 3 into a navigable page and route structure. The application distinguishes between public routes, authentication routes, and protected routes.

The public area contains the landing page, legal pages, archive search, and conversion workflow. These routes communicate the purpose of QDArchive and provide low-friction access to its main public functionality. The authentication area contains sign-in, sign-up, forgot-password, and reset-password pages. The protected area contains the dashboard, project creation, authenticated project details, project and file management, versioning, downloads, sharing controls, and user settings.

The information architecture is illustrated in Figure 4.2. Unlike the user-journey diagram in Chapter 3, which describes the progression of user activities, this figure describes the structural organization of pages and route areas.

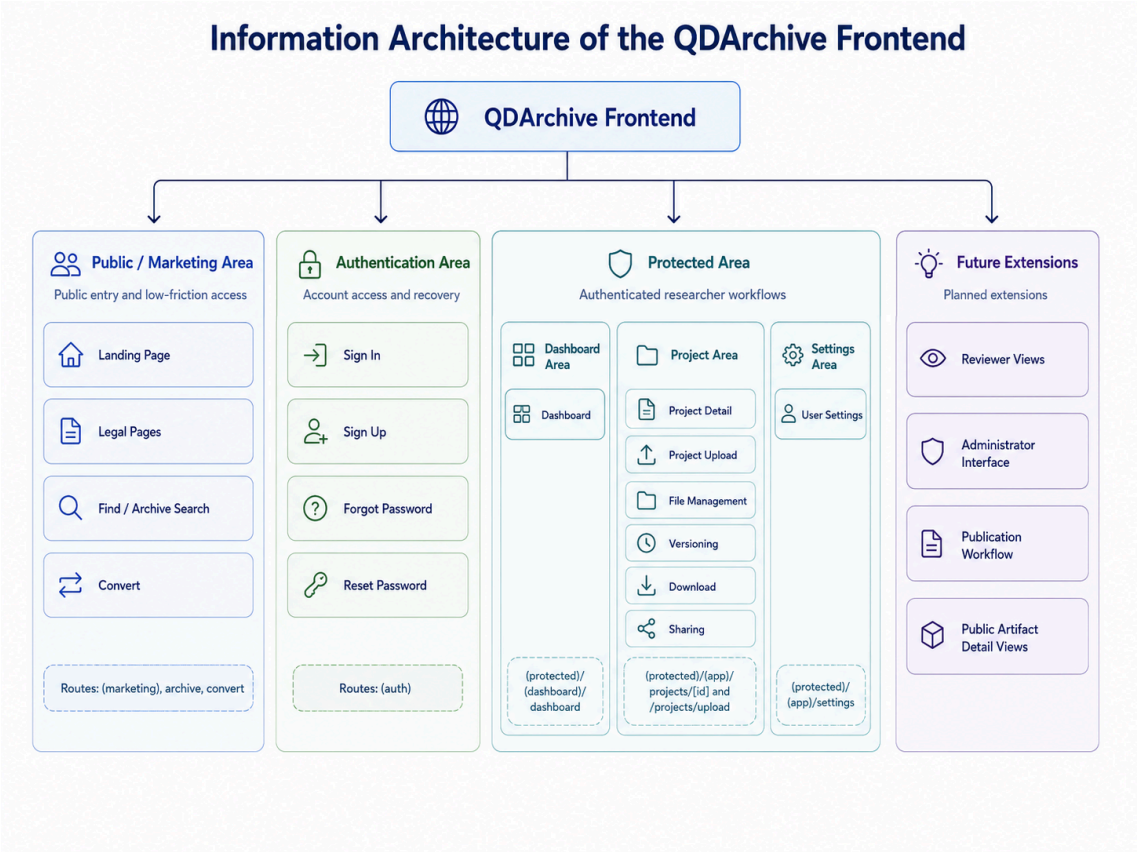


Figure 4.2: Information architecture of the QDArchive frontend

The route names shown in the architecture figures and summarized in Table 4.3 correspond to the final application route structure.

Route area	Example routes	Design purpose and current scope
Public / marketing area	`(marketing)`, landing page, legal pages	Introduces QDArchive, presents its main solution areas, provides legal information, and guides users toward *Find*, *Convert*, sign in, or sign up.

Archive area	`archive`
	`archive/[id]`
	`archive/secure`
		Provides archive search and route locations intended for public and secure artifact access. Search is implemented, while the complete recipient-facing project views remain incomplete.

Convert area	`convert`	Provides file upload, source- and target-format selection, conversion initiation, feedback, result download, and browser-local conversion history.

Authentication area	`(auth)/sign-in`
	`(auth)/sign-up`
	`(auth)/forgot-password`
	`(auth)/reset-password`
		Supports account creation, authentication, and password recovery.

Protected dashboard area	`(protected)/(dashboard)/dashboard`	Provides the authenticated entry point, recommended actions, and access to recent and existing projects.
Protected project area	`(protected)/(app)/projects/[id]` `(protected)/(app)/projects/upload`	Supports guided project creation, authenticated project details, metadata and file management, missing-file completion, versioning, downloads, and owner-facing access controls.
Protected settings area	`(protected)/(app)/settings`	Allows authenticated users to manage profile information and profile images and displays explicitly marked future account functionality.

Table 4.3: Main route areas and their design purpose in the QDArchive frontend

The architecture preserves extension points for reviewer views, administrator interfaces, publication workflows, domain-specific artifact inspection, notifications, and further account operations. These additions can be introduced without restructuring the existing public, authentication, and registered-researcher route areas.

4.4 User Experience Design

The user experience design focuses on presenting complex QDA artifact workflows through guided interactions, clear hierarchy, and visible system states. Because a project may contain an analysis file, multiple referenced primary files, generic supplementary files, metadata, versions, missing references, and access settings, the interface cannot represent every artifact as a single ordinary file.

The design therefore separates related operations into focused pages, panels, steps, and dialogs. It communicates what information has already been supplied, which files remain missing, which actions are currently available, and which functionality is not available in the prototype.

A central UX decision was to provide low-friction public access. Visitors can open the landing page, search the archive, and use conversion before creating an account. Registration is introduced when a user wishes to maintain persistent projects or access authenticated project-management functionality.

The dashboard acts as the main authenticated entry point. It provides recommended actions for finding public projects, creating a project, and opening the user's existing projects. This reduces the number of decisions immediately after login and provides direct access to the most important researcher workflows. Planned features should not be displayed as apparently active dashboard actions unless they are clearly labelled as unavailable.

The guided project-creation workflow is divided into three steps:

1. entry of project metadata;
2. upload and format handling for the main analysis artifact; and
3. review before final creation.

This structure applies progressive disclosure by presenting only the information required at the current stage. The review step allows users to confirm the project name, description, tags, uploaded file, detected format, and default private visibility before submission.

File upload is supported through drag-and-drop and manual file selection. Where the format can be detected, the detected label is shown to the user. If detection fails, the user is prompted to select the appropriate source format manually. This provides a recoverable path instead of treating an undetected file as an unrecoverable error.

The authenticated project detail page is designed as the main structured representation of a QDArchive project. It separates project metadata from file information and distinguishes between the analysis artifact, referenced primary files, and generic supplementary files. When referenced files are absent, the interface displays a warning and identifies the affected file entries. Users can then upload missing files or edit supported file metadata.

Direct file-viewing controls are present, but uploaded files do not currently open reliably. The implemented project-detail design should therefore be understood primarily

as a file-organization and project-management interface rather than a complete browser-based content inspection environment.

The landing page contains supervisor-approved product copy describing capabilities such as inspecting codes, memos, source material, and media in the browser. These statements represent the longer-term QDArchive product vision. They do not describe fully operational functionality in the evaluated prototype. Domain-specific inspection of analytical structures and reliable browser-based viewing of uploaded files remain future work.

To preserve readability in print, the principal UX examples are shown in two separate figures rather than combining four complete screenshots into a single small composite. Figure 4.3 illustrates progressive disclosure in project creation, while Figure 4.4 illustrates structured file presentation and missing-file guidance.

The prototype uses recurring interaction patterns across its workflows:

- Empty states explain when no projects, files, or conversions are available and direct users toward the next action.
- Validation messages explain missing or invalid input close to the relevant form field.
- Disabled actions prevent submission while mandatory input is missing.
- Explicit unavailability labels distinguish future functionality from temporarily disabled operational actions.
- Toast notifications confirm successful operations and report request failures.
- Confirmation dialogs are used for consequential actions such as deletion, visibility changes, and secure-link regeneration.
- Warning banners identify incomplete artifact states such as missing referenced files.
- Badges communicate formats, file states, visibility, and other compact status information.

Toast notifications are initiated by frontend callbacks after a request succeeds or fails. The backend or server-side route returns a response, while the relevant query or mutation callback determines how that outcome is communicated to the user. This distinction is important when describing the later implementation data flow.

Responsive behavior was addressed through adaptive layouts and component breakpoints. Dashboard cards, metadata panels, forms, and file sections can use multiple columns on wider screens and stack vertically on narrower viewports. Navigation is reduced to a more compact structure on mobile sizes. Tables and dialogs are adjusted where necessary to prevent horizontal overflow.

The responsive design supports desktop, tablet, and representative mobile viewport sizes. However, the prototype was primarily developed and evaluated through desktop browsers and browser-based viewport emulation. The design should therefore not be interpreted as having undergone comprehensive physical-device or cross-browser validation.

Accessibility was addressed through readable typography, visible labels, contrast-conscious design, consistent button states, validation feedback, and use of established Material UI components. A complete accessibility audit was not performed. Future work should include systematic evaluation against WCAG 2.2, covering keyboard navigation, focus order,

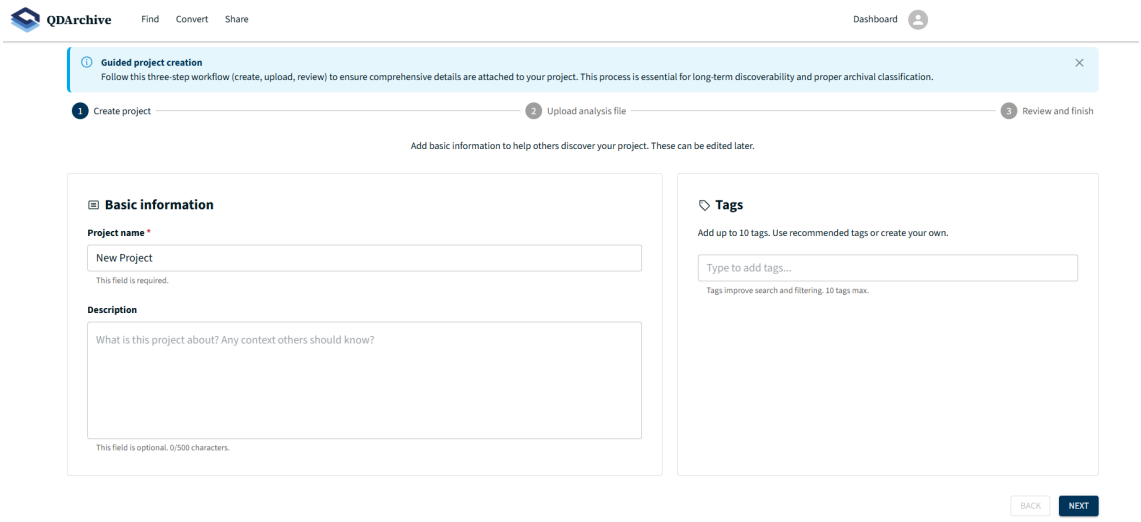


Figure 4.3: Guided project creation workflow with separated metadata, file-upload, and review stages

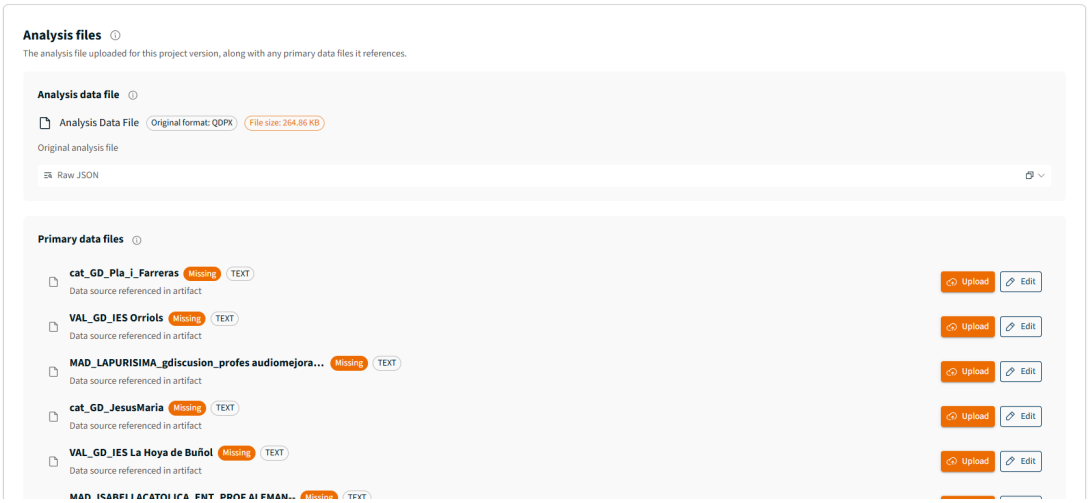


Figure 4.4: Authenticated project detail view with structured file sections and missing-file guidance

focus trapping, semantic structure, screen-reader output, form errors, colour contrast, and responsive zoom behavior (World Wide Web Consortium, 2024).

4.5 QDArchive Visual Identity

The QDArchive visual identity was designed to communicate clarity, stability, and trust. Since the application concerns research artifacts, file conversion, access conditions, and project management, the interface uses a restrained visual language rather than decorative or highly animated presentation.

The design uses a light theme, clear typography, rounded cards and panels, subtle borders, consistent iconography, and status indicators. The visual system is implemented through shared design tokens, a centralized Material UI theme, and supporting Tailwind utility classes.

The design tokens define the principal palette, typography, font sizes, weights, spacing values, border radii, and layout measurements. These values are mapped into the Material UI theme and selected component overrides. Tailwind utilities are used for supplementary layout and spacing concerns but do not replace the centralized component theme.

The main design tokens are summarized in Table 4.4.

Design element	**Token or value**	**Design role**
Primary colour	`#003459`	Used for principal actions, headings, navigation emphasis, and the main QDArchive brand tone.
Secondary colour	`#007ea7`	Provides supporting emphasis and variation within the blue palette.
Tertiary colour	`#00a8e8`	Used for lighter accents and selected interaction highlights.
Dark colour	`#00171f`	Provides the main text colour and high-contrast interface elements.
Light colour	`#ffffff`	Provides the principal page, card, and panel background.
Grey colour	`#c1c3c7`	Supports borders, inactive states, secondary information, and neutral interface elements.
Heading font	`Source Serif 4`	Provides headings with an academic and publication-oriented character.
Body font	`Source Sans 3`	Supports readability in body text, labels, forms, tables, cards, and file lists.
Spacing scale	`0, 4, 8, 16, 24, 32, 40, 48, 56, 64, 72, 80`	Provides consistent spacing between sections, cards, fields, actions, and interface groups.
Border radius	`4, 8, 10, 12, 50`	Supports consistent shapes for controls, cards, panels, badges, and circular elements.

Table 4.4: Main visual identity tokens used in the QDArchive frontend

Typography establishes the principal content hierarchy. *Source Serif 4* is used for headings and gives the interface a research-oriented visual character. *Source Sans 3* is used for body text, labels, navigation, form content, tables, and file information. The limited weight range supports consistency and avoids unnecessary visual variation.

Colour is used for both identity and communication. The primary dark blue connects the interface to the QDArchive brand and is used for principal actions and headings. Neutral greys support borders, placeholders, inactive states, and secondary information. Status colours are used to communicate success, warning, error, missing-file, and unavailable states.

Warnings and unavailable functionality must remain visually distinct. A warning indicates an operational condition that the user may be able to resolve, such as a missing referenced file. An unavailable state indicates that the operation is outside the current prototype. The latter should therefore include explicit explanatory text rather than relying only on a warning colour or disabled button.

Cards and panels group related information throughout the interface. Dashboard cards present principal actions and recent projects. Project-creation panels separate metadata, tags, upload, and review information. Project-detail panels separate metadata, file roles, missing references, versions, downloads, and sharing controls.

Phosphor Icons provides consistent iconography for upload, download, search, navigation, files, folders, settings, profile actions, sharing, and warning states. Icons are paired with labels wherever the meaning may otherwise be ambiguous.

The current prototype uses a light theme. The centralized Material UI palette and the supporting Tailwind utility layer provide a technical basis for a later dark theme. However, dark mode was not implemented and would require additional design work and testing for contrast, status colours, diagrams, images, dialogs, tables, and uploaded-content presentation.

4.6 Summary

This chapter explained how the requirements from Chapter 3 were translated into the system design of the QDArchive frontend. The design goals focused on making QDA artifact workflows understandable, lowering the entry barrier for visitors, introducing complexity progressively, and clearly distinguishing implemented functionality from future functionality.

The application architecture combines browser-executed React components with server-executed Next.js routes. Routes, feature components, React Query hooks, query keys, services, contexts, typed models, utilities, and theme configuration are separated by responsibility. The final route structure was verified against the implemented codebase.

Material UI provides the principal component system and authoritative visual theme, while Tailwind CSS supports general utility styling for layout, spacing, sizing, and responsive adjustments. QDConvert and Archiver are application dependencies used on the server side rather than browser-side frontend libraries. QDConvert is maintained outside the QDArchive repository and is invoked through server-side application routes.

The information architecture separates public, authentication, and protected researcher workflows. Public routes support onboarding, legal information, archive search, and conversion. Protected routes support the dashboard, guided project creation, authenticated

project details, file management, versioning, downloads, access configuration, and user settings.

The user experience design emphasizes guided workflows, validation, visible project and file states, missing-file guidance, explicit unavailable states, responsive layouts, and consistent feedback. Landing-page statements concerning browser-based inspection of codes, memos, and media represent the longer-term QDArchive vision rather than fully implemented prototype functionality.

The visual identity combines centralized design tokens, Material UI theming, supporting Tailwind utilities, serif headings, sans-serif interface text, restrained colours, cards, panels, badges, and consistent iconography.

Together, these decisions provide the design basis for the implementation discussed in Chapter 5. The next chapter explains how the routes, components, services, server-side integrations, state-management mechanisms, and principal workflows were implemented.

5 Implementation

This chapter describes how the frontend design presented in Chapter 4 was realized in the QDArchive prototype. Whereas the preceding chapter focused on architectural, navigational, interaction, and visual design decisions, this chapter focuses on their implementation in the codebase. It explains the implemented scope, source-code organization, data-access mechanisms, authentication and state management, public workflows, authenticated project workflows, reusable interface patterns, and known implementation limitations.

The implementation is organized around the three user-facing solution areas introduced in Chapter 3: *Find*, *Convert*, and *Share*. In the frontend implementation, *Share* is understood broadly and includes authenticated project creation, project and file management, versioning, downloading, visibility control, and secure-link management. In addition to these areas, the prototype includes public entry pages, email/password authentication, a researcher dashboard, profile settings, and shared interface components.

The application was implemented using Next.js, React, TypeScript, Material UI, Tailwind CSS, TanStack React Query, and Supabase-related libraries. It follows the architectural separation described in Figure 4.1: route-level components provide page entry points, feature components implement workflows, React Query hooks coordinate server state, service modules perform API communication, and shared contexts provide authentication and application-wide feedback.

This chapter distinguishes between an *implemented mechanism* and *verified behavior*. Descriptions of components, hooks, service functions, routes, and request flows are based on inspection of the implementation. A statement that a workflow operated successfully refers only to behavior that was exercised during the manual evaluation described in Chapter 6. This distinction is particularly important for file viewing and recipient-facing sharing, where controls and request mechanisms exist but the complete user-facing behavior is not operational.

5.1 Implementation Scope

The implementation scope was determined by the requirements in Chapter 3, the frontend design in Chapter 4, and the functionality supported by the available backend and server-side services. The main objective was to support the visitor and registered-researcher workflows required to find, convert, create, manage, version, download, and configure access to QDA projects. Reviewer-specific and administrator-specific interfaces remained outside the completed prototype.

The public entry layer is implemented through the landing page, legal-information pages, and navigation to *Find*, *Convert*, sign in, and sign up. Email/password authentication includes registration, login, logout, forgot-password, reset-password, authentication initialization, and session handling. Google sign-in is visually represented in parts of the interface but is treated as a separate future requirement rather than as part of the implemented email/password authentication workflow.

The *Find* area is partially implemented. Visitors can access archive search and browse public projects, while authenticated users can additionally select their own projects. Basic text search, sorting, public-project presentation, authenticated project filtering, loading feedback, errors, and empty states are implemented. Complete public artifact detail views, file-level search, and advanced metadata filtering remain unavailable.

The *Convert* area is implemented as an end-to-end workflow for guest and authenticated users. It supports file selection, source-format detection, manual source-format selection when detection fails, target-format selection, acceptance of the Terms of Service, conversion progress feedback, converted-file download, and browser-local guest conversion history. Guest users are offered optional account creation after conversion. For authenticated users, the conversion result is associated with a private project where supported by the server-side workflow.

Authenticated project management represents the largest part of the implementation. It includes the dashboard, project list, guided project creation, authenticated project detail pages, project metadata editing, semantic separation of file roles, missing-file completion, supplementary-file management, downloads, project versioning, public/private visibility changes, and owner-facing secure-link management. Project, file, and version deletion operations use explicit confirmation dialogs.

Several workflows remain incomplete. The dashboard contains a non-operational starred-projects area. Notification settings, account-data export, and account deletion are unavailable. Public and secure links can be configured by the project owner, but the recipient-facing pages that should resolve those links do not display the project. File-viewing controls are present, but uploaded primary and supplementary files do not open reliably. Reviewer and administrator interfaces are not implemented.

For reproducibility, the implementation described in this thesis corresponds to the QDArchive release published on 29 May 2026. The source code is maintained in a private repository on GitHub.[2] The evaluated release is identified by the tag `v-1.0.0` and commit `56d44b7`. Because the repository is not publicly accessible, access may be granted to the examiners upon request and subject to approval by the QDArchive project maintainers.

The implementation coverage is summarized in Table 5.1. These labels describe whether frontend functionality is present in the prototype; they are not the evaluation statuses used in Chapter 6.

Table 5.1: Implementation coverage of the QDArchive frontend prototype

Area	**Implemented frontend functionality**	**Status**
Public entry and legal pages	Landing page, feature presentation, navigation to the principal solution areas, footer navigation, Privacy Policy, Cookie Policy, Terms of Service, Contact/Imprint, and Third-Party Notices.	Implemented
Email/password authentication	Registration, login, logout, forgot-password, reset-password, authentication initialization, session handling, and authenticated navigation.	Implemented
External identity provider	Google sign-in is visually represented but is not configured or connected to an authentication provider.	Not implemented

Continued on next page

[2] https://github.com/QDArchive/qdarchive

Table 5.1 – continued from previous page

Area	Implemented frontend functionality	Status
Find and archive search	Public archive access, text search, public-project listing, authenticated *My projects* filtering, sorting, loading and error feedback, and empty states. Advanced filters, file-level search, and complete public artifact detail views are unavailable.	Partial
Convert	Guest and authenticated conversion, file upload, format detection, manual source-format selection, target-format selection, Terms of Service acceptance, conversion feedback, download, optional registration, and browser-local recent-conversion history.	Implemented
Dashboard	Recommended actions, access to *Find*, project creation, project browsing, and recent-project presentation. The starred-projects area remains non-operational.	Partial
Project creation	Three-step workflow for project metadata, analysis-file upload, validation, format identification, review, private-by-default creation, and redirection to the created project.	Implemented
Project details and metadata	Authenticated project detail page, version selection, creation date, access type, source type, uploader information, editable project name, description, and tags.	Implemented
Project file management	Separation of analysis, primary, and supplementary files; missing-file identification and upload; supplementary-file upload; file metadata editing; download and deletion actions; and a raw JSON inspection aid. File viewing remains defective.	Partial
Project download	Selection of export format, optional operating-system variant, analysis-only or all-files scope, and ZIP or TAR.GZ archive format where applicable.	Implemented
Project versioning	Creation of new versions, retention or replacement of the analysis file, selective transfer of supplementary files, version switching, version metadata editing, and version deletion.	Implemented
Sharing and visibility	Public/private visibility changes, confirmation warnings, public-link generation, and secure-link generation, copying, disabling, enabling, and regeneration. Recipient-facing public and secure project views remain incomplete.	Partial

Continued on next page

Table 5.1 – continued from previous page

Area	Implemented frontend functionality	Status
User settings	Profile information and profile-image management. Notifications, account-data export, and account deletion are represented as unavailable functionality.	Partial
Reviewer and administrator interfaces	Reviewer views, review queues, moderation controls, and administrator interfaces are not present in the prototype.	Not implemented

5.2 Project Structure and Code Organization

The codebase is organized using a feature-oriented structure. Responsibilities are separated into route definitions, domain-specific components, shared components, query hooks, query-key definitions, service modules, contexts, models, utilities, configuration, and styling. This structure follows the architecture introduced in Figure 4.1 and the route organization summarized in Table 4.3.

The Next.js `app` directory provides the route-level structure. Route groups distinguish public and marketing pages, authentication pages, dashboard pages, and authenticated application pages. Route components remain comparatively small and delegate most workflow behavior to components from feature directories. This prevents page definitions from accumulating detailed domain logic.

Feature components are grouped by functional area, including archive search, authentication, conversion, dashboard, marketing, projects, and settings. Larger workflows are decomposed further into components responsible for specific interaction areas.

The conversion workflow, for example, is coordinated by the `ConvertView` component together with the `useConvertFlow` hook. `ConvertView` composes the visible upload, format-selection, consent, processing, completion, and history areas. `useConvertFlow` manages temporary workflow state and connects those interface components to conversion, format, project, and download operations. This division keeps rendering concerns separate from the workflow state and asynchronous actions.

The project area follows a similar compositional approach. The principal `ProjectDetails` component manages the active tab and selected project version and delegates rendering to the detailed project area and Files Overview. The detailed area is composed from the project header, metadata sections, analysis and primary files, and supplementary files. Files Overview reuses file sections and cards to present the selected version's files in a more compact, action-oriented form.

Components used in more than one feature and not owned by one domain are placed in `components/common`. Examples include the shared checkbox field, empty-state presentation, and file-upload step. Domain-specific components remain within their feature directories even when they are internally composed from smaller reusable components.

Server-state logic is separated from the visual hierarchy. Query hooks are grouped by domain under `queries/hooks`, while query-key factories are stored under `queries/keys`. Service modules under `services` contain lower-level API requests. Components can there-

fore depend on hooks such as `useProject`, `useUserProjects`, or `useUploadVersionFiles` without constructing request URLs or controlling cache invalidation directly.

TypeScript models and data-transfer objects describe projects, versions, files, uploaded-file requests, download configuration, share links, user roles, conversion states, and other exchanged data. Shared formatting, download, error, and file helpers are placed in utility modules. Theme configuration and global design values remain in the styling area described in Section 4.5.

The principal source-code areas are summarized in Table 5.2.

Code area	**Primary responsibility**	**Examples**
`app`	Defines Next.js routes, layouts, route groups, page-level entry points, and server-side API routes.	Public pages, authentication routes, dashboard routes, project routes, settings routes, and API handlers.
`components`	Contains feature-specific interface components and workflow composition.	Archive, auth, convert, dashboard, marketing, projects, and settings.
`components/common`	Contains interface components reused across multiple feature areas and not owned by one domain.	`CheckboxField`, `EmptyState`, and shared `UploadStep`.
`queries/hooks`	Provides domain-oriented React Query hooks for retrieving and modifying server data.	Project, version, file, conversion, user, and share-link queries and mutations.
`queries/keys`	Defines stable query-key factories used to identify, update, and invalidate cached data.	Project keys, version keys, file keys, and share-link keys.
`services`	Encapsulates HTTP requests and lower-level communication with application API routes.	`ProjectService`, `FileService`, `ConvertService`, and `DownloadService`.
`contexts`	Provides application-wide client state and cross-cutting functionality.	Authentication, Supabase client access, and toast feedback.
`models` and `types`	Defines TypeScript models, DTOs, enums, and shared state types.	Projects, versions, files, share links, download requests, roles, and conversion states.
`utils` and `config`	Contains shared helper functions and centralized configuration.	Formatting, error extraction, file extensions, download helpers, supported formats, and size limits.
`styles`	Contains theme setup and application-wide visual configuration.	Material UI theme, typography, palette, spacing, and component overrides.

Table 5.2: Main source-code areas and responsibilities in the QDArchive implementation

This organization reduces direct coupling between interface components and request details. A component can initiate an operation through a mutation hook without needing

to know how the request body is serialized, how an API error is parsed, or which cached queries must be refreshed. Conversely, service modules remain independent of the visual components that consume them.

5.3 Data Fetching and Backend Integration

The frontend communicates with QDArchive functionality through a layered integration structure. Components call domain-specific query or mutation hooks. These hooks invoke service functions, which send requests to Next.js API routes. The server-side route then communicates with the relevant backend, storage, conversion, or archive-generation functionality.

Responses conceptually return through the same layers: the server-side route produces a response, the service parses it, the query or mutation hook updates React Query state, and the consuming component re-renders. User feedback such as a toast is triggered by frontend query or mutation callbacks after the response is interpreted; it is not emitted directly by the backend.

The flow is illustrated in Figure 5.1. The diagram is intentionally simplified and represents the principal request, response, cache, and feedback relationships rather than every internal function call.

5.3.1 Service Layer and API Requests

Service modules encapsulate request URLs, HTTP methods, request serialization, and response parsing. For example, `ProjectService` provides functions for retrieving the authenticated user's projects, retrieving an individual project, updating project metadata, deleting a project, and creating a project from an uploaded file.

Standard JSON bodies are used for metadata updates. Project creation combines structured metadata with a binary analysis file and therefore uses `FormData`. The project DTO is serialized as JSON and appended together with the file. The service checks the response status and delegates unsuccessful responses to the shared `throwApiError` helper. An abbreviated version is shown in Listing 5.1.

```
export const createProjectFromFile = async (
  projectDTO: ProjectDTO,
  file: File,
): Promise<Project> => {
  const formData = new FormData();
  formData.append("project", JSON.stringify(projectDTO));
  formData.append("file", file);
  const response = await fetch("/api/projects", {
    method: "POST",
    body: formData,
  });
  if (!response.ok) {
    await throwApiError(response,"Failed to create project");
  }
  return response.json();
};
```

Listing 5.1: Service method for creating a project from metadata and an uploaded file in `ProjectService.ts`

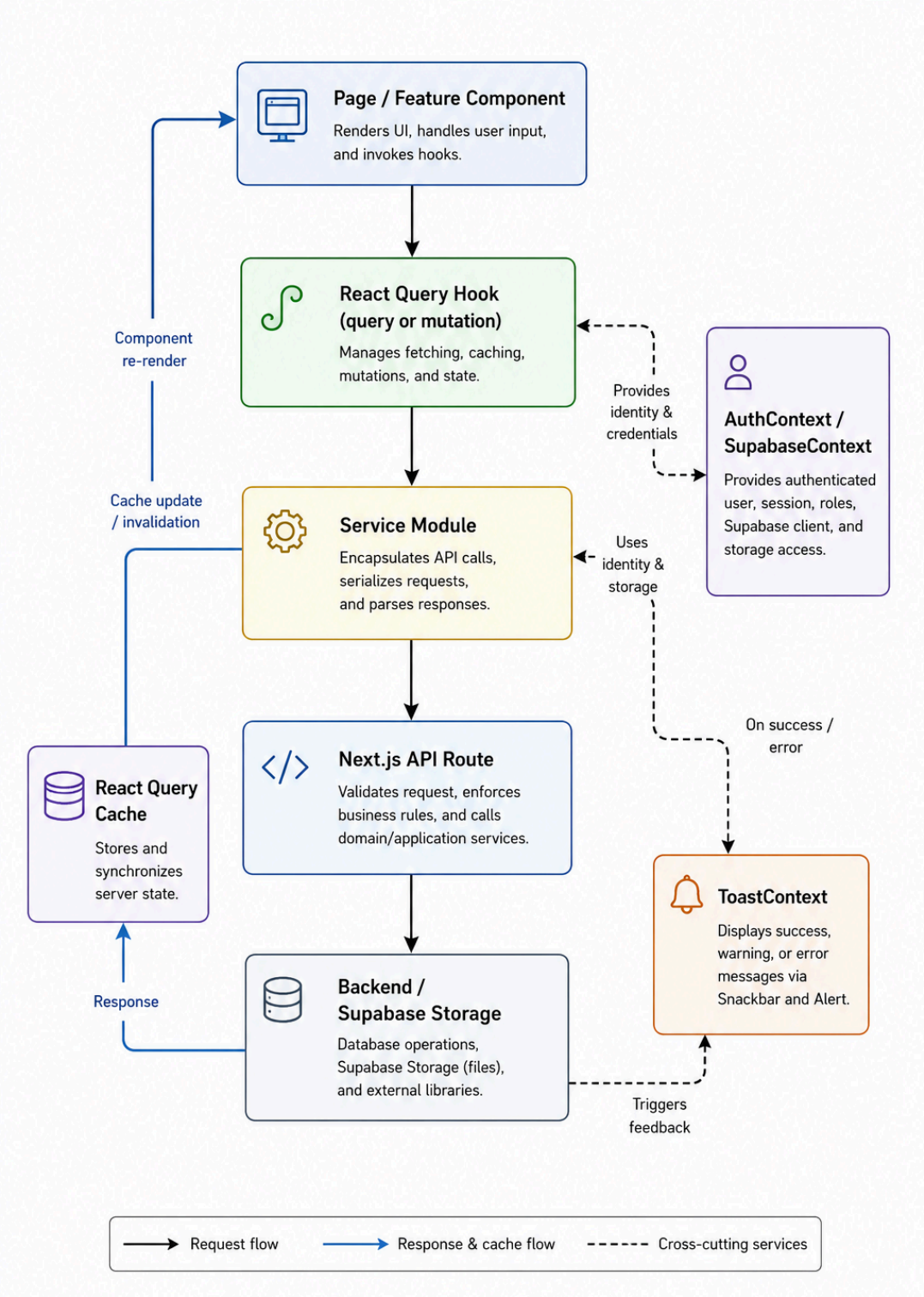


Figure 5.1: Simplified request, response, cache-synchronization, and frontend-feedback flow

The same structure is used for other operations. Retrieval uses `GET`, metadata modification uses `PATCH`, and deletion uses `DELETE`. Centralizing these operations avoids repeated request construction inside page and dialog components.

File operations use project-version-file metadata together with Supabase Storage information. The intended viewing mechanism retrieves or derives the stored-file path and attempts to create a URL that the browser can open. The download mechanism retrieves the binary content and passes it to the browser as a downloadable file.

Manual testing verified that stored files could be downloaded. It did not verify successful in-browser viewing: view actions currently fail to open uploaded primary and supplementary files reliably. The presence of URL-construction code and view buttons is therefore not treated as proof that file viewing works.

5.3.2 Queries, Mutations, and Cache Synchronization

TanStack React Query manages remote data and asynchronous operations. Read operations are represented as query hooks, while creation, update, upload, deletion, conversion, versioning, and sharing operations are represented as mutation hooks.

Project query hooks use centralized query keys. The `useUserProjects` hook identifies the user's project list through a user-specific key and remains disabled until a user identifier is available. The `useProject` hook follows the same approach for an individual project. A short stale period allows recently retrieved data to be reused while still being refreshed.

Mutation hooks execute write operations and synchronize cached data afterward. The project-creation mutation shown in Listing 5.2 stores the created project in its individual cache entry and invalidates the authenticated user's project collection.

```
export function useCreateProjectFromFile(
  userId: string | null,
) {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: ({
      projectDTO,
      file,
    }: {
      projectDTO: ProjectDTO;
      file: File;
    }) => createProjectFromFile(projectDTO, file),

    onSuccess: async (createdProject) => {
      queryClient.setQueryData(
        projectsKeys.project(createdProject.id),
        createdProject,
      );

      if (userId) {
        await queryClient.invalidateQueries({
          queryKey: projectsKeys.userProjects(userId),
        });
      }
    },
  });
}
```

Listing 5.2: Project-creation mutation and React Query cache synchronization

The same cache-oriented approach is used elsewhere. Project metadata updates refresh project data, file mutations refresh version-file queries, version mutations refresh version lists and selected versions, and share-link mutations refresh sharing data. Central query-key factories prevent cache-key arrays from being duplicated throughout the application.

The project detail page coordinates several queries. It retrieves the project, latest project version, version list, optionally selected historical version, and files belonging to the effective version. The effective version is derived from either the user's explicit selection or the latest version. The resulting project, version, and file data are passed to the detailed and compact file views.

5.3.3 Authentication and Session State

Supabase authentication is integrated through two related contexts. `SupabaseProvider` creates a browser client using the configured public Supabase URL and anonymous key and makes it available to descendant components.

`AuthProvider` manages the authenticated user, session, user identifier, role, initialization state, and logout operation. Authentication state is stored in a reducer so that initial session resolution and sign-out transitions are handled explicitly. The provider registers an authentication-state listener before requesting the current session, reducing the possibility of missing an event during initialization.

When a stored session exists, the provider validates the authenticated user before treating the application as initialized. The context exposes `user`, `session`, `role`, and the derived role flags `isAdmin`, `isReviewer`, and `isUser`. Reviewer and administrator pages are not implemented, but the role-aware context provides an extension point for later role-specific routing and components.

Protected components wait until `isAuthInitialized` is true before rendering account-owned data. Project lists and detail pages display loading feedback during session resolution and prevent unauthenticated access to protected project operations.

The verified email/password workflow includes registration, sign-in, sign-out, password-reset request, password reset, session persistence after refresh, and protected-route handling. Google sign-in was not configured and is evaluated separately rather than reducing the status of the implemented email/password workflow.

5.3.4 Client State, Guest Persistence, and Feedback

Different state mechanisms are used according to the lifetime and responsibility of the data. These mechanisms are summarized in Table 5.3.

State category	**Mechanism**	**Examples**
Local interface state	React `useState`, `useMemo`, `useReducer`, and component callbacks	Dialog visibility, selected tabs, current workflow step, selected version, form values, drag state, and temporary file selections.
Remote server state	TanStack React Query queries and mutations	Projects, project versions, version files, user profiles, export formats, and share links.
Authentication state	`SupabaseContext` and `AuthContext`	Supabase browser client, user, session, role, initialization state, and logout.
Guest conversion history	Browser `localStorage` through a custom hook	Up to ten recent guest conversions, including source file, target format, download name, and creation time.
Transient global feedback	`ToastContext` using Material UI `Snackbar` and `Alert`	Success, information, warning, and error messages after uploads, updates, deletions, conversions, and validation failures.

Table 5.3: Frontend state-management mechanisms and responsibilities

The guest conversion workflow stores a limited history in `localStorage`. The custom hook reads stored records when the conversion page is initialized and writes new records after successful guest conversions. The history is capped at ten entries. It can remain available in the same browser profile until cleared or replaced and should therefore not be described as session-only storage.

Error handling is divided between service-level error construction and interface-level presentation. Service functions convert unsuccessful responses into exceptions. Queries can display persistent inline errors with retry actions, while mutations commonly use the global toast mechanism for immediate feedback.

`ToastProvider` exposes `showToast`, `showSuccess`, and `showError`. Toasts use Material UI `Snackbar` and `Alert`, appear in the lower-right corner, and close automatically after 3.5 seconds. The toast is triggered in frontend callbacks after the request outcome is received and interpreted.

5.4 Implementation of Public User Flows

The public part of the application provides access to QDArchive before authentication. It includes the landing page, legal pages, authentication entry points, archive search, and conversion. These routes implement the low-entry-barrier principle described in Section 4.1.

5.4.1 Public Entry and Authentication

The landing page introduces the purpose of the service and presents *Find*, *Convert*, and *Share*. Navigation directs visitors to archive search, conversion, sign in, and account creation. Additional sections describe the broader product vision and link to legal information.

Some landing-page copy describes future browser-based inspection of codes, memos, source material, and media. As explained in Chapter 4, these statements represent the longer-term QDArchive vision rather than completed prototype behavior. Reliable file viewing and domain-specific analytical inspection are not implemented.

The legal area contains the Privacy Policy, Cookie Policy, Terms of Service, Contact/Imprint, and Third-Party Notices. These pages remain accessible before a user uploads data or creates an account.

Authentication pages implement email/password registration, sign-in, forgot-password, and reset-password workflows. Registration collects basic profile information and requires acceptance of the Terms of Service and Privacy Policy. Supabase handles authentication and password recovery, while the frontend provides form validation, loading states, and error feedback. Google sign-in is visible but unavailable.

5.4.2 Archive Search

The archive interface provides text search for project names, descriptions, and keywords. Public projects are displayed as cards containing project metadata, tags, source information, and dates where available. Sorting options allow results to be ordered by date or name.

The interface distinguishes between publicly visible projects and projects owned by the authenticated user. If a visitor selects *My projects*, the frontend presents a sign-in prompt rather than requesting private-project data.

Loading, empty, and error states are represented separately. An empty result suggests changing the query or available filters, while a failed request displays an actionable error state. Manual evaluation verified basic project-level search, sorting, public/private result separation, and the authenticated *My projects* scope. File-level search, advanced filtering, and public artifact detail access remain unavailable.

5.4.3 Conversion Workflow

The conversion workflow is implemented through the collaboration of `ConvertView` and `useConvertFlow`. `ConvertView` composes the visible upload, format-selection, consent, processing, completion, registration, and history states. The `useConvertFlow` hook stores the selected file, detected source format, selected target format, Terms of Service acceptance, optional account-creation preference, conversion status, result information, and

active guest-history tab. It also connects the interface to format, conversion, project, and download queries and mutations.

Users can select a file through the browser or by dragging it into the upload area. The upload component checks the maximum accepted size and attempts to identify the source format. When detection fails, the interface opens a manual source-format dialog. The selected file and format can be removed or replaced before submission.

Available target formats are loaded from the conversion-related service. The selector excludes or disables the current source format as a target. Before conversion begins, the user must accept the Terms of Service. The conversion action remains disabled until the required file, source format, target format, and acceptance state are available.

After submission, the interface displays a processing state and prevents duplicate conversion actions. A successful response changes the workflow to a completion state and enables result download. For authenticated users, the response can also provide access to the created private project.

Guest users can download the converted file without registration, proceed to account creation, or reset the workflow. Successful guest conversions are added to browser-local history, limited to ten entries. Each stored record contains information such as the original filename, source and target formats, creation time, and download-again action.

The behavior for guests and authenticated users is summarized in Table 5.4.

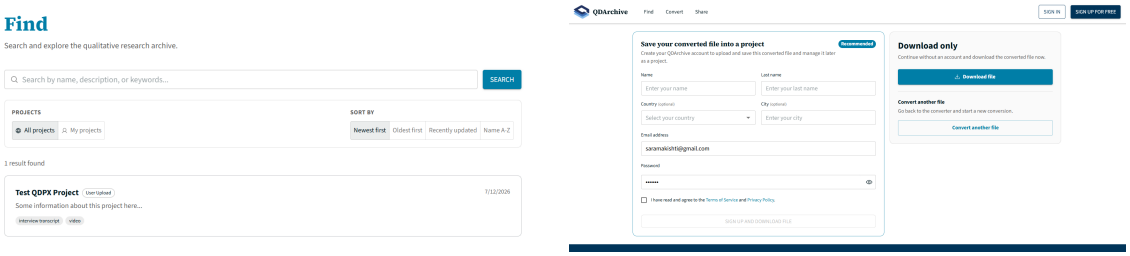
Workflow aspect	**Guest user**	**Authenticated user**
Access	Conversion is available without registration after acceptance of the Terms of Service.	Conversion is available through the authenticated session.
Created project	A private processing project may be created and associated with the guest conversion workflow.	A private project is associated with the registered user's account.
Completion options	Download the result, optionally create an account, or convert another file.	Download the result or navigate to the associated project.
Recent conversions	Up to ten entries are retained in browser-local storage.	Conversion-created projects are available through the authenticated project area.
Persistence	History depends on the current browser profile and can be cleared locally.	Projects are persisted as account-owned QDArchive projects.

Table 5.4: Behavior of the conversion workflow for guest and authenticated users

To retain legibility in print, Figure 5.2 shows two representative public-workflow states. Additional intermediate states are included in Appendix B.

5.5 Implementation of Authenticated User Flows

Authenticated workflows extend the public functionality with persistent project ownership and project-management operations. They include the dashboard, project list, guided



(a) Archive search results

(b) Successful guest conversion

Figure 5.2: Representative implemented states of the public Find and Convert workflows

project creation, project metadata and file management, downloads, versioning, visibility configuration, secure-link management, and profile settings.

5.5.1 Dashboard and Project Access

After authentication, the dashboard acts as the main entry point for researcher workflows. It provides actions for finding public projects, creating a project, and browsing the user's projects. Recent projects are displayed to reduce the number of navigation steps required to resume work.

The complete project collection is implemented through the `MyProjects` component. Data are loaded using the authenticated user identifier and a user-project query. Loading, error, and empty states are displayed separately. When no projects exist, the shared empty-state component directs the user to project creation.

Projects can be displayed in table or grid form. Both representations expose the project name, visibility state, modification information, navigation to the project detail page, and project deletion. Deletion requires confirmation and refreshes the affected queries after success.

The dashboard includes a planned starred-projects section. Persistent starring and retrieval are not implemented. The area should therefore be presented with an explicit *Not available yet* state rather than appearing operational.

5.5.2 Guided Project Creation

Project creation is implemented as a three-step workflow:

1. entering project metadata;
2. uploading the analysis file; and
3. reviewing and creating the project.

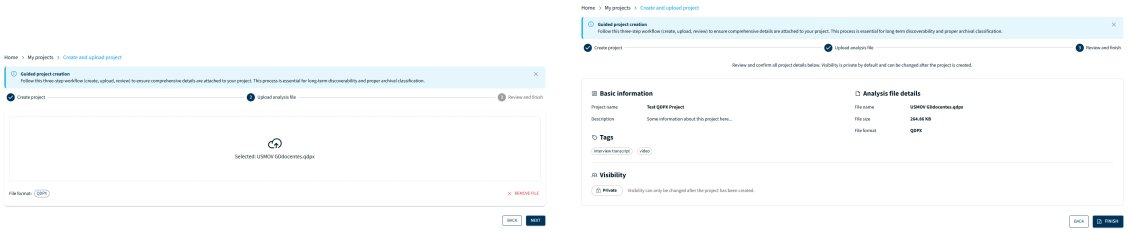
The metadata step requires a project name and supports an optional description and up to ten tags. Suggested tags are combined with free-text entry, and duplicate values are prevented.

The upload step uses the shared upload component. It supports drag-and-drop and browser selection, checks the file-size limit, attempts format detection, and provides manual format selection if detection fails. The user cannot continue until a file and source format have been established.

The review step presents the collected metadata, tags, filename, detected format, file size, and default private visibility. This makes the intended access state visible before submission.

On completion, the workflow constructs a project data transfer object (DTO) containing the metadata, authenticated user identifier, private state, and source information. The DTO and analysis file are submitted through the creation mutation. Following a successful response, the upload state is cleared, a success toast is displayed, and the user is redirected to the authenticated project detail page.

To improve print readability, Figure 5.3 shows the analysis-file and final-review stages. The metadata-entry state is available in the appendix.



(a) Detected analysis file (b) Review before creation

Figure 5.3: Implementation of the guided project-creation workflow

5.5.3 Project Details and Version Selection

The project detail page coordinates project metadata, the latest version, the complete version list, an optionally selected historical version, and files belonging to the effective version. The effective version is either the version selected by the user or the latest version.

The component maintains the active tab and selected version. The retrieved project name is also passed to the breadcrumb presentation so that the route identifier can be replaced by a readable title.

Because the project name is loaded asynchronously, the raw project identifier may be shown briefly in the breadcrumb before the readable title becomes available. This visual flash does not prevent navigation but is a known presentation defect.

If the selected version references files that have not yet been uploaded, the page displays a warning containing the number of missing files. The action scrolls to the primary-file section so that the user can continue with the relevant upload.

The page contains two principal tabs. *Project details* presents metadata and file-management functionality in a detailed layout. *Files overview* presents analysis, primary, and supplementary files through collapsible sections and compact action cards.

5.5.4 Project Metadata Editing

The project header contains the title, selected version, download action, sharing action, and additional project operations. The metadata area displays the creation date, access type, project source, uploader, description, and tags.

Project-name editing is performed directly in the header. The title can be replaced by an input field, validated, and submitted through the project-update mutation. Description

editing similarly provides edit, save, and cancel actions without requiring navigation to a separate page.

Tags can be added from suggested values or entered manually and can be removed individually. The maximum of ten tags is also enforced during editing.

Long project titles, descriptions, tags, source values, or other metadata can overflow or wrap poorly in some layouts, particularly in constrained widths. This does not prevent metadata editing but reduces readability and is recorded as an implementation limitation.

5.5.5 Analysis, Primary, and Supplementary Files

Files are categorized according to their role rather than displayed as one undifferentiated list. The analysis file represents the principal structured artifact for the selected version. Primary files represent source material referenced by the analysis project, such as transcripts, PDFs, images, audio, or video. Supplementary files represent additional material such as codebooks, licences, consent documents, or notes.

The analysis-file section displays its filename, original format, optional size, and description. The parsed data associated with the selected project version can also be expanded as formatted raw JSON. This raw JSON presentation is a developer-oriented technical inspection aid for confirming that structured data were received and associated with the version. It is not a completed QDA artifact viewer and does not provide domain-specific navigation of codes, coded segments, memos, annotations, or analytical relationships.

Primary files are displayed with names, descriptions, formats, and missing-file states. Existing metadata can be edited through a dialog. For a missing reference, the upload dialog shows the expected filename and format. If the selected extension differs, the interface displays a warning but permits continuation.

Supplementary files can be uploaded individually or in groups. The multi-file review dialog allows each file to be renamed, described, removed from the pending upload, and expanded for metadata entry. Empty display names prevent submission. Existing supplementary files can subsequently be edited or deleted.

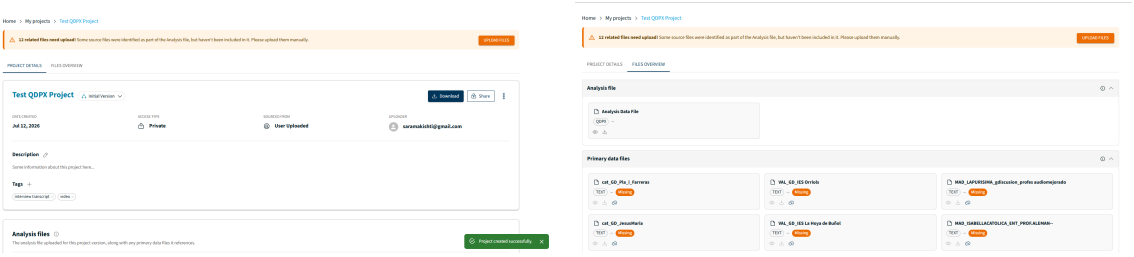
Files Overview uses reusable cards for available and missing files. Missing files do not expose view or download operations. Missing primary files expose an upload action, and supplementary files expose deletion where applicable.

Download behavior was verified during manual testing. File-viewing behavior was not: view actions in both the detailed project area and Files Overview fail to open uploaded primary and supplementary files correctly. The implementation therefore supports file organization, metadata management, upload, and download, but not reliable browser-based file inspection.

5.5.6 Project Download

The download action opens a configuration dialog for the selected project version. Export formats are retrieved dynamically. The user selects the desired output format and, where required, an operating-system variant.

Users can choose an analysis-only download or include all files associated with the version. An all-files package contains the analysis file, uploaded primary files, and supplementary files. The archive can be generated as ZIP or TAR.GZ. The frontend constructs a request



(a) Project details with missing-file warning

(b) Categorized Files Overview

Figure 5.4: Implemented project detail and file-management views

containing the version identifier, export format, operating-system option, file scope, and archive type.

Manual evaluation verified analysis-only and all-files downloads as well as ZIP and TAR.GZ packaging in the tested environment.

5.5.7 Project Versioning

Project versioning is accessible through the project actions menu. A new version requires a name and may contain a description. The user chooses whether to retain the current analysis file or replace it.

Supplementary files can be transferred independently. The user may copy all, copy none, or choose specific files. The selected options are serialized into the version DTO and submitted with an optional replacement analysis file.

After successful creation, the interface displays feedback and selects the new version. The selector lists available versions, identifies the latest version, and allows switching between them. Version names and descriptions can be edited. Versions can be deleted while at least one version remains, and the interface selects another version if the active version is removed.

5.5.8 Visibility and Sharing

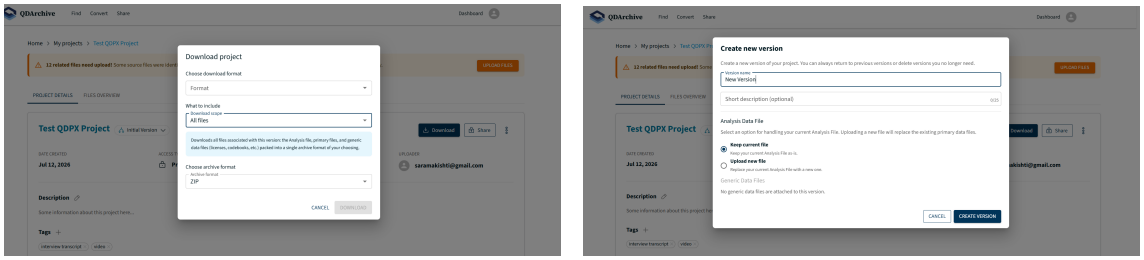
Sharing is configured through a dialog opened from the project header. The visibility area represents the current private or public state. Both private-to-public and public-to-private transitions require confirmation because they affect the intended access conditions.

For a public project, the dialog displays a public URL and copy action. For a private project, the interface supports secure-link generation and management. A secure link can be copied, disabled, re-enabled, or regenerated.

These mechanisms were verified only from the owner's perspective. Visibility changes and secure-link management operations worked during testing. The generated public and secure links did not resolve to working recipient-facing project pages and instead led to a not-found state. Owner-facing access configuration must therefore be distinguished from completed sharing or publication.

Changing a project to public also does not constitute a complete publication workflow. Licence selection, metadata review, possible approval steps, and a working public artifact detail page remain unavailable.

To keep the screenshots readable, download and versioning are presented together in Figure 5.5, while sharing is shown separately in Figure 5.6.



(a) Download configuration (b) Creation of a new project version

Figure 5.5: Implemented project download and versioning workflows

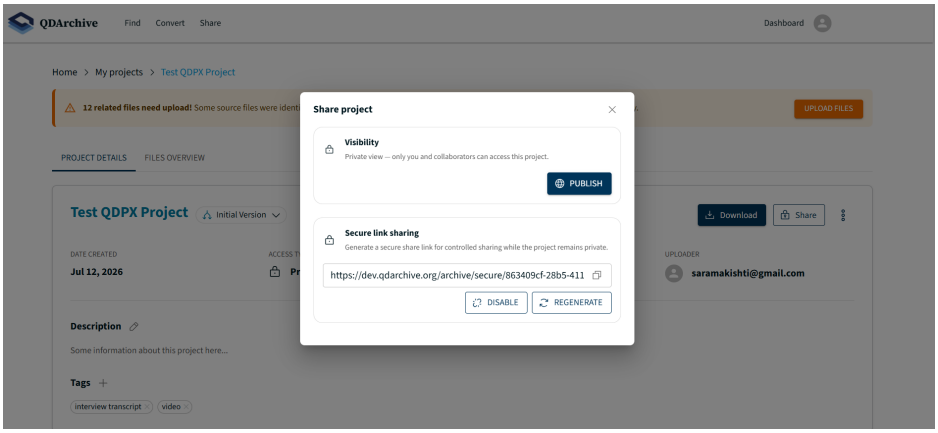


Figure 5.6: Owner-facing project visibility and secure-link controls

5.5.9 User Settings

The settings area allows authenticated users to update their first name, last name, country, city, and profile image. The registered email address is displayed but is not directly editable.

The settings interface also establishes locations for notification preferences, account-data export, and account deletion. These operations are not connected to completed backend functionality and should be accompanied by an explicit *Not available yet* state.

5.6 Reusable UI Components and Feedback Patterns

Material UI provides most interface primitives, including buttons, dialogs, form controls, cards, tables, alerts, tabs, steppers, accordions, chips, snackbars, and progress indicators. These components are customized through the centralized theme described in Section 4.5.

The frontend also defines shared QDArchive components for patterns occurring across multiple feature areas. Components used by at least two features and not owned by one domain are placed in `components/common`. Domain-specific reusable components remain within their feature directory.

Component or pattern	Implementation role	Example usage
`CheckboxField`	Presents a checkbox with a title, description, and optional required marker.	Terms of Service acceptance and optional conversion-related preferences.
`EmptyState`	Displays an icon, title, description, and optional primary action when no data are available.	Empty project lists and recent-conversion views.
Shared `UploadStep`	Provides drag-and-drop and browser selection, selected-file feedback, format state, manual format selection, and removal.	Guided project creation and QDA file-upload workflows.
`FileSection` and `FileCard`	Provide collapsible semantic file groups and compact file-action cards.	Analysis, primary, and supplementary files in Files Overview.
Confirmation dialogs	Require confirmation before destructive or access-changing operations.	Deleting projects, files, or versions and changing visibility.
Status chips and alerts	Communicate formats, missing files, visibility, validation warnings, and persistent errors.	Format identification, missing-file status, access state, and failed queries.
Unavailable-state presentation	Distinguishes planned features from active but temporarily disabled controls.	Starred projects, notifications, data export, and account deletion.
`ToastContext`	Provides transient success, information, warning, and error feedback.	Project creation, uploads, metadata updates, deletion, sharing, and version creation.

Table 5.5: Reusable interface components and feedback patterns

The shared upload component centralizes the visual drop area, drag state, hidden file input, file removal, format hint, format chip, and manual-selection action. Feature hooks remain responsible for workflow-specific validation and submission.

Empty states are treated as active interface elements rather than blank spaces. Where appropriate, they provide a direct action such as creating the first project or returning to conversion.

Long-running operations display progress and disable duplicate submission. Persistent query failures use inline alerts with retry actions. Destructive operations use confirmation dialogs. Successful mutations commonly use transient toasts.

These patterns separate feedback by duration and importance. Inline alerts remain visible when action is required, confirmation dialogs interrupt high-risk operations, progress indicators communicate pending work, unavailable states identify prototype boundaries, and toasts acknowledge completed operations.

5.7 Implementation Limitations

The prototype covers the principal visitor and researcher workflows, but it does not represent the complete QDArchive product vision. Some requirements are only partially implemented, and some interface mechanisms partially implemented, and some interface mechanisms exist without producing the intended end-to-end behavior.

The principal limitations are summarized in Table 5.6.

Table 5.6: Main limitations of the frontend implementation

Area	**Current limitation**	**Effect on the prototype**
Find	File-level search, advanced metadata filters, and complete public project detail views are unavailable.	Visitors can discover public projects through basic text search but cannot complete the full discovery and inspection workflow.
Public project access	Public URLs can be generated, but the unauthenticated artifact view is not operational.	Owners can change visibility, but recipients cannot open the generated public project page.
Secure sharing	Secure links can be generated and managed, but recipient-facing secure-project resolution is not operational.	Owner-facing link management works, while controlled access cannot be completed by the recipient.
Named-user sharing	Inviting specified users or collaborators by email is not implemented.	Sharing is limited to owner-facing visibility and link controls.

Continued on next page

Table 5.6 – continued from previous page

Area	Current limitation	Effect on the prototype
Publication workflow	Open-licence selection, metadata review, approval steps, and a working public artifact view are unavailable.	Changing a project to public does not satisfy the complete publication requirement.
External authentication	Google sign-in is not configured.	Users authenticate through the implemented email/password workflow.
Dashboard	Starred projects are represented only as planned functionality.	Users can access recent and owned projects but cannot maintain a favourites collection.
Account settings	Notifications, account-data export, and account deletion are not implemented.	The corresponding areas must be presented explicitly as unavailable rather than operational.
Reviewer and administrator roles	Reviewer interfaces, administrator moderation pages, and review queues are unavailable.	The prototype primarily supports visitors and registered researchers.
File viewing	View actions are present in Project Details and Files Overview, but uploaded primary and supplementary files fail to open reliably.	Users can manage metadata and download stored files but cannot inspect their contents directly in the frontend.
Raw artifact JSON	Serialized artifact data can be expanded and copied, but no domain-specific analytical visualization is implemented.	The JSON area supports technical inspection only and is not a completed viewer for codes, coded segments, memos, annotations, or relationships.
Breadcrumb rendering	The route identifier may be displayed briefly before asynchronous project metadata replaces it with the project name.	Users can momentarily see an internal identifier when opening or refreshing a project page.
Metadata overflow	Long project names, descriptions, tags, source values, or other metadata can wrap or overflow poorly in constrained layouts.	Some project information becomes harder to read, particularly at smaller viewport widths.

Continued on next page

Table 5.6 – continued from previous page

Area	Current limitation	Effect on the prototype
Accessibility verification	Accessibility-oriented design decisions were applied, but no complete WCAG audit or assistive-technology evaluation was performed.	Accessibility compliance cannot be claimed from the implementation alone.
Theme support	The centralized theme can support later extension, but dark mode is not implemented.	The prototype is evaluated only in its light theme.

These limitations reflect the incremental nature of the project. The implementation prioritizes the main visitor and registered-researcher workflows and provides structures into which additional functionality can be integrated. Public and secure project pages can reuse parts of the existing project presentation with different access controls, while reviewer and administrator functionality can be introduced through additional protected routes and feature modules.

Chapter 6 evaluates the verified behavior systematically against the functional and non-functional requirements. Chapter 7 then prioritizes the work needed to complete recipient access, file inspection, publication, role-specific interfaces, accessibility validation, and technical quality assurance.

6 Evaluation

This chapter evaluates the implemented QDArchive frontend against the functional and non-functional requirements defined in Chapter 3. The evaluation examines whether the required frontend behavior is present, whether representative workflows can be completed from beginning to end, and to what extent the implementation satisfies the broader quality goals established for the prototype.

The evaluation is based primarily on requirements inspection, manual scenario validation, implementation inspection, and formative feedback from QDArchive project supervisors and developers. It is therefore an implementation-oriented and formative evaluation rather than a formal usability study with representative users. The evaluation does not attempt to demonstrate production readiness or statistically generalizable user acceptance. Instead, it determines how closely the evaluated prototype realizes the consolidated frontend requirements and identifies the remaining functional and quality-related gaps.

6.1 Evaluation Approach

The QDArchive frontend was developed iteratively while the broader project requirements and available backend functionality continued to evolve. The evaluation therefore uses the consolidated requirement baseline presented in Chapter 3. As explained in Section 3.1, this baseline was established at the end of the implementation period and contains implemented, partially implemented, and deferred requirements. The evaluation does not derive the requirements retrospectively from the observed implementation.

Four complementary forms of evidence were used:

1. **Requirements inspection:** the frontend was compared with the functional and non-functional requirements defined in Chapter 3.
2. **Manual scenario validation:** representative visitor and authenticated-researcher workflows were executed from beginning to end in the deployed development environment.
3. **Project-supervisor and developer feedback:** the deployed prototype was reviewed through recorded supervisor feedback rounds, weekly project discussions, and informal testing by QDArchive developers.
4. **Implementation inspection:** relevant routes, components, hooks, services, contexts, validation rules, and feedback mechanisms were inspected to determine how the observed frontend behavior was implemented.

The evaluation distinguishes between implementation evidence and verified behavior. The presence of a route, button, service method, or request mechanism does not by itself demonstrate that the complete user-facing workflow operates successfully. For example, owner-facing secure-link controls exist, but the recipient-facing secure route does not display the shared project. Similarly, file-view actions are present, but uploaded files do not open reliably. These cases are therefore evaluated as partial or unfulfilled rather than being considered complete from implementation inspection alone.

No formal study involving representative qualitative researchers, repository users, reviewers, or administrators was conducted. The project supervisors and developers were already familiar with QDArchive's goals, terminology, and implementation context. Their feed-

back is treated as expert project feedback and implementation evidence, not as evidence of general usability or user acceptance.

6.1.1 Status Categories

Each evaluated requirement or capability is assigned one of the three status categories defined in Table 6.1. The status columns throughout this chapter use only these categories. Qualifications concerning the tested environment, user role, available evidence, or remaining limitation are stated separately in the evidence or assessment text.

Status	Meaning in this evaluation
Fulfilled	The required frontend behavior is present and was successfully exercised during manual testing within the stated evaluation environment. Any limitations of the tested formats, devices, scenarios, or user roles are stated separately.
Partially fulfilled	A meaningful part of the requirement is implemented, but at least one required interaction, state, access mode, user role, or validation condition remains incomplete, defective, or insufficiently verified.
Not fulfilled	The required frontend behavior is unavailable in the evaluated prototype or is represented only by a placeholder, unavailable route, or non-operational control.

Table 6.1: Status categories used in the frontend evaluation

6.1.2 Evaluation Environment

The main evaluation was conducted in May and June 2026 using the deployed QDArchive development environment. The frontend and backend were exercised together through the deployed application, while selected route, state-management, and service mechanisms were also inspected in the source code.

The frontend version to which the results in this chapter refer is identified by the release tag and commit recorded in Table 6.2. A separate backend commit identifier was not recorded as part of this frontend evaluation.

Navigation, dialogs, project cards, status indicators, forms, tables, and upload areas were inspected at the tested viewport sizes. The evaluation did not reveal a general breakdown of the responsive layouts, but the metadata-overflow defect described in Chapter 5 remained visible in some constrained layouts. Responsive testing relied primarily on browser emulation rather than a broad collection of physical devices.

6.1.3 Feedback-Driven Iteration

Two dedicated project-supervisor feedback rounds were documented through recorded walkthrough videos. The platform was also reviewed during weekly project discussions and tested informally by three additional QDArchive developers. Findings were converted into internal GitHub Projects tickets where appropriate and were incorporated into the implementation according to their priority and feasibility.

This process was formative. Feedback was used to revise the prototype during development rather than being collected only after implementation had ended. It therefore provides

Evaluation aspect	Environment or method
Deployed environment	https://dev.qdarchive.org/
Evaluated frontend release	QDArchive release from 29 May 2026, tag `v-1.0.0`, commit `56d44b7`.
Main evaluation period	May and June 2026.
Primary browser	Google Chrome.
Desktop operating systems	Windows and macOS.
Responsive testing	Desktop and laptop layouts together with Chrome device emulation for representative mobile and tablet viewports.
Frontend and backend	Exercised together in the deployed development environment; selected frontend mechanisms were additionally inspected in the codebase.
Tested conversion labels	Selected `.qdpx`, `.refi`, and `.qda` inputs and targets as exposed by the integrated conversion service. The terminology follows the clarification in Section 3.4.3; `.refi` and `.qdpx` should not be interpreted as unrelated standards.
Automated testing	No unit, integration, or browser-based end-to-end test suite was implemented.
Accessibility tools	No Lighthouse, axe, WAVE, screen-reader, or comparable formal accessibility audit was performed.

Table 6.2: Environment and methods used to evaluate the frontend prototype

evidence of design iteration and expert project review, but not of usability performance among representative target users.

Examples of resulting changes are summarized in Table 6.3.

Feedback area	Resulting frontend change
Project creation was not sufficiently clear as a single workflow	The process was reorganized into a three-step guided workflow for metadata, analysis-file upload, and final review.
Landing-page content and priorities required clarification	The content hierarchy was revised to communicate the purpose of QDArchive and the *Find*, *Convert*, and *Share* entry points more clearly.
Dashboard actions required clearer prioritization	The dashboard was reorganized around recommended actions for finding public projects, creating a project, and browsing existing projects.
Project details contained a large amount of complex information	The presentation was revised repeatedly and divided into metadata, version controls, analysis files, primary files, supplementary files, and a separate Files Overview tab.

Table 6.3: Examples of frontend changes resulting from project-supervisor and developer feedback

6.2 Functional Requirements Coverage

The functional evaluation follows the requirement groups defined in Tables 3.4 to 3.9. To make the relationship between the requirements and the evaluation explicit, each row in Table 6.4 identifies the corresponding requirement IDs.

Requirements are grouped only when they belong to the same capability and received the same evaluation status. Requirements from the same functional area are separated when their outcomes differ. This prevents implemented functionality from hiding an unfulfilled requirement within the same broader workflow.

Table 6.4: Evaluation of functional requirement coverage

ID(s)	Requirement area	Status	Evaluation evidence and remaining gap
FR-PE1–FR-PE2	Landing page and public navigation	Fulfilled	The landing page was available without authentication and introduced QDArchive and its main workflows. Navigation to *Find*, *Convert*, *Share*, sign in, and sign up was available. The content organization was revised through the project-supervisor and developer feedback rounds.

Continued on next page

Table 6.4 – continued from previous page

ID(s)	Requirement area	Status	Evaluation evidence and remaining gap
FR-PE3	Legal pages	Fulfilled	The Privacy Policy, Cookie Policy, Terms of Service, Contact/Imprint, and Third-Party Notices were accessible through the public navigation or footer.
FR-PE4–FR-PE6	Email/password registration and authentication	Fulfilled	Registration, required acceptance of the Terms of Service and Privacy Policy, login, logout, invalid-credential feedback, forgot-password, reset-password, protected-route handling, and session persistence after refresh were manually exercised successfully. Email verification was not required, and expired reset links were not tested.
FR-PE7	External identity provider	Not fulfilled	Google sign-in was represented visually but was not configured or connected to an external authentication provider.
FR-F1–FR-F8	Basic Find and archive search	Fulfilled	The public Find page was accessible without authentication. Search was exercised using project names, descriptions, tags, and keywords. The implemented sorting options, result metadata, public/private project separation, authenticated *My projects* access, sign-in prompt, and empty states behaved as expected.
FR-F9–FR-F10	Detailed artifact access and advanced Find filters	Not fulfilled	Search results could not be opened in a working public artifact detail view. Advanced filtering by artifact type, licence, file format, and other metadata was not implemented.

Continued on next page

Table 6.4 – continued from previous page

ID(s)	Requirement area	Status	Evaluation evidence and remaining gap
FR-C1–FR-C10, FR-C12	Guest conversion workflow	Fulfilled	Guest users could access conversion, select or drag and drop a supported file, inspect or correct the detected source format, remove the selected file, choose a valid target, accept the Terms of Service, run conversion, and download the result without creating an account. Submission was prevented while mandatory input was missing and the source format could not be selected as its own target. Unsupported files and failed conversion states produced feedback. Testing did not cover every possible source-target combination or malformed internal structure.
FR-C11	Guest conversion history	Fulfilled	Recent guest conversions were retained in browser-local storage and presented in the recent-conversions view. Repeated download was exercised, and the history was limited to ten entries.
FR-C13	Post-conversion account creation	Fulfilled	A guest could complete conversion and proceed to account creation without first losing access to the conversion result.
FR-C14	Authenticated conversion and project association	Fulfilled	Conversion was completed while authenticated, and the resulting private project became available through the authenticated project area.
FR-PM1–FR-PM3	Dashboard and project access	Fulfilled	The authenticated dashboard provided recommended actions for finding public projects, creating a project, and browsing the user's projects. Recent projects and access to the full project collection were available. Loading, empty, and error states were also present.
FR-PM4	Starred projects	Not fulfilled	The dashboard contained a planned starred-projects area, but users could not create or retrieve a persistent collection of favourite projects.

Continued on next page

Table 6.4 – continued from previous page

ID(s)	Requirement area	Status	Evaluation evidence and remaining gap
FR-PM5–FR-PM12	Guided project creation	Fulfilled	The three-step workflow for metadata, analysis-file upload, and review was completed manually. Project-name validation, description and tag input, file-size and format handling, manual format selection, private-by-default visibility, creation feedback, and redirection to the created project were exercised successfully.
FR-PM13	Project detail metadata	Fulfilled	The project detail page displayed the project title, selected version, creation date, access type, source, uploader, description, and tags. The breadcrumb could briefly display the internal project identifier before the title loaded, but the required metadata remained available.
FR-PM14	File-role separation	Fulfilled	The frontend distinguished between the analysis file, referenced primary files, and supplementary files in both the detailed project view and Files Overview.
FR-PM15–FR-PM17	Missing primary files	Fulfilled	Projects containing missing references displayed a warning and identified the affected files. Uploading a missing primary file was completed successfully.
FR-PM18	Project and file metadata editing	Fulfilled	Project names, descriptions, and tags could be edited. The names and descriptions of supported primary and supplementary files could also be changed through mutation-backed dialogs. Long values could wrap or overflow poorly in constrained layouts, but the editing operations remained available.

Continued on next page

Table 6.4 – continued from previous page

ID(s)	Requirement area	Status	Evaluation evidence and remaining gap
FR-PM19	File overview, inspection, and management	Partially fulfilled	The file-centred overview was implemented, and users could manage primary and supplementary files. However, view actions in both the Project Details view and Files Overview failed to open uploaded files correctly. The raw JSON area provided only a technical inspection aid and not a domain-specific artifact viewer.
FR-PM20	Project versioning	Fulfilled	New versions were created while retaining or replacing the analysis file. Supplementary files could be copied using all, none, or selected-file options. Version selection, editing, latest-version indication, deletion, and fallback after deletion were also exercised.
FR-PM21–FR-PM22	Project download	Fulfilled	Analysis-only and all-files downloads completed successfully. Users could select an export format, file scope, operating-system variant where applicable, and ZIP or TAR.GZ packaging.
FR-PM23	Secure share-link access	Partially fulfilled	Project owners could generate, copy, disable, enable, and regenerate secure links. The generated recipient-facing secure URL led to a 404 page, so the controlled-access workflow could not be completed by the recipient.
FR-PM24	Public/private visibility control	Fulfilled	Private-to-public and public-to-private transitions were completed through confirmation dialogs and produced the expected owner-facing feedback. This status applies only to visibility control and does not imply completion of publication or public recipient access.
FR-PM25	Public artifact detail view	Not fulfilled	A public URL could be generated, but it led to a 404 page and did not render the public project metadata, files, or download options.

Continued on next page

Table 6.4 – continued from previous page

ID(s)	Requirement area	Status	Evaluation evidence and remaining gap
FR-US1–FR-US5	Profile settings	Fulfilled	Authenticated users could access settings, view and edit supported profile fields, and upload, change, or remove a profile image. The registered email address was shown as non-editable account information.
FR-US6–FR-US8	Notification, export, and account-deletion settings	Not fulfilled	Notification preferences, account-data export, and account deletion were represented as planned or unavailable functionality without completed frontend and backend operations.
FR-DA1	Reviewer-oriented artifact inspection	Not fulfilled	A dedicated reviewer interface and recipient-facing inspection workflow were not implemented. Secure links could be managed by project owners, but recipients could not open and inspect the shared project.
FR-DA2	Named-user sharing	Not fulfilled	Inviting specific registered users or collaborators by email was not implemented.
FR-DA3	Complete publication workflow	Partially fulfilled	Projects could be marked public and could appear in public search results, but open-licence selection, metadata review, a complete publication process, and a working public artifact detail page were not implemented.
FR-DA4–FR-DA5	Administrator interfaces and moderation	Not fulfilled	Administrator review queues, ingestion approval, user administration, project-status management, and moderation controls were outside the completed prototype.

The strongest functional coverage was achieved in conversion, guided project creation, metadata management, missing-file completion, versioning, and project download. The main partially fulfilled or unfulfilled requirements concerned recipient-facing public and secure access, uploaded-file inspection, advanced archive discovery, complete publication, named-user sharing, account-management extensions, and reviewer or administrator workflows.

6.3 Non-Functional Requirements Assessment

The non-functional requirements in Table 3.10 cannot all be verified through one successful interaction. They were assessed through interface inspection, manual workflow execution, implementation structure, responsive inspection, and formative project feedback.

Because no representative-user study, automated accessibility audit, broad device study, or production-quality test suite was conducted, the statuses remain limited to the evidence available from the prototype evaluation.

Table 6.5: Assessment of the frontend non-functional requirements

ID	Quality	Status	Assessment
NFR-1	Usability	Partially fulfilled	The workflows use guided steps, clear actions, disabled invalid submissions, confirmation dialogs, and status feedback. Project-supervisor and developer feedback resulted in several usability-related revisions. However, no formal usability study with representative qualitative researchers was conducted.
NFR-2	Learnability	Partially fulfilled	The landing page, dashboard, recommended actions, explanations, helper text, and guided project creation support initial understanding. Learnability was not measured through observation of independent first-time users.
NFR-3	Transparency	Fulfilled	The frontend communicates private-by-default creation, required acceptance of the Terms of Service, detected formats, missing files, visibility state, download scope, version state, pending operations, and destructive consequences. Recipient-facing limitations and unavailable functionality are documented separately.
NFR-4	Privacy awareness	Partially fulfilled	Projects are private by default, conversion requires acceptance of the Terms of Service, and visibility changes require confirmation. However, recipient-facing access remains incomplete, account deletion and export are unavailable, and no formal privacy-compliance assessment was conducted.

Continued on next page

Table 6.5 – continued from previous page

ID	Quality	Status	Assessment
NFR-5	Low entry barrier	Fulfilled	*Find* and *Convert* are available without registration. Guests can download a converted result without creating an account, while registration is offered as an optional continuation.
NFR-6	Responsiveness	Partially fulfilled	Desktop and laptop layouts and representative mobile and tablet viewports were inspected in Chrome. Navigation, dialogs, cards, tables, and upload areas generally adapted to the available width. Testing relied mainly on emulation, did not provide broad physical-device or cross-browser coverage, and metadata overflow remained visible in some constrained layouts.
NFR-7	Accessibility	Partially fulfilled	Material UI provides established form and dialog components, and the interface uses labels, validation states, and selected keyboard behavior. No screen-reader test, systematic keyboard-only walkthrough, automated audit, or complete contrast and focus assessment was performed.
NFR-8	Feedback and error handling	Fulfilled	Loading indicators, disabled states, inline alerts, retry actions, confirmation dialogs, and global toast messages are used across queries and mutations. Invalid credentials, unsupported files, failed conversion, missing consent, and other selected failure states were manually exercised.
NFR-9	Consistency	Fulfilled	Material UI, the centralized theme, shared typography, common components, repeated file patterns, status chips, confirmation dialogs, and the toast mechanism provide consistent interaction and visual behavior.

Continued on next page

Table 6.5 – continued from previous page

ID	Quality	Status	Assessment
NFR-10	Interoperability communication	Fulfilled	The frontend displays detected source formats, supports manual source-format selection, prevents identical source and target selection, loads target labels dynamically, reports missing referenced files, and exposes format and archive options during download.
NFR-11	Maintainability	Fulfilled	The codebase separates routes, feature components, common components, query hooks, query keys, services, contexts, models, utilities, and theme configuration. TypeScript models and reusable components reduce duplication and isolate responsibilities. The absence of automated regression tests remains a significant maintenance risk but does not remove the implemented structural separation required by NFR-11.
NFR-12	Extensibility	Partially fulfilled	Route groups, feature-oriented components, service boundaries, dynamic format loading, typed models, and role-aware authentication provide extension points. Reviewer, administrator, recipient-facing sharing, artifact-inspection, and advanced-search extensions have not yet been implemented to validate this extensibility through completed functionality.

The quality goals most strongly demonstrated by the prototype are low entry barrier, transparency, consistent interaction patterns, feedback and error handling, interoperability communication, and structural maintainability. Usability, learnability, accessibility, responsiveness, privacy awareness, and extensibility received supporting implementation evidence, but the available evaluation does not justify claiming that they have been comprehensively validated.

6.4 End-to-End Workflow Validation

Individual components can appear to satisfy their immediate purpose while the combined workflow still fails. Representative end-to-end scenarios were therefore executed manually in the deployed development environment.

The scenarios in Table 6.6 form the most complete consolidated record of the manual validation performed for this thesis. A separate formal test protocol or exportable GitHub

test-ticket appendix was not produced during execution. This limits later reproducibility and is acknowledged in Section 6.5.

Table 6.6: Manual end-to-end workflow validation

Scenario	**Actions performed**	**Observed result**	**Status**
Authentication and protected access	Register, log in, refresh the page, log out, attempt protected access while unauthenticated, request a password reset, and set a new password.	The session persisted after refresh, logout removed authenticated access, protected routes redirected appropriately, and password recovery completed successfully. Expired reset links were not tested.	Fulfilled
Public discovery	Search by project name, description, tags, and keywords; apply each implemented sorting option; compare public and private visibility; and open *My projects* while authenticated.	Matching public projects appeared, private projects remained absent from public results, sorting behaved as expected, and authenticated users could access their own projects. Public artifact details remained unavailable.	Partially fulfilled
Guest conversion	Open *Convert* without authentication, upload a supported file, resolve source-format detection, choose a target, test missing acceptance, convert, download, inspect recent conversions, and download again.	The conversion and repeated-download workflow completed successfully. Unsupported-file and failed-conversion states produced feedback.	Fulfilled
Guest registration after conversion	Complete conversion as a guest, choose account creation, register through the completion state, and retain access to the conversion result.	The registration continuation was available, and the converted result remained accessible for download.	Fulfilled
Authenticated project creation	Sign in, enter project metadata and tags, upload an analysis file, review the project, and complete creation.	A private project was created, success feedback appeared, and the browser navigated to the authenticated project detail page.	Fulfilled
Missing-file completion	Open a project with missing references, follow the warning action, select a missing primary file, upload it, and update its metadata.	The missing reference was identified, and the upload completed successfully with feedback.	Fulfilled

Continued on next page

Table 6.6 – continued from previous page

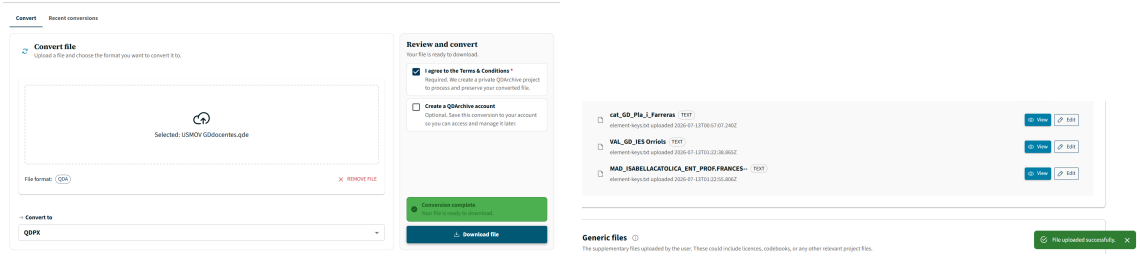
Scenario	Actions performed	Observed result	Status
Supplementary-file management	Upload multiple supplementary files, edit pending names and descriptions, submit them, and then edit and delete an uploaded file.	Multi-file upload, metadata editing, and deletion completed successfully.	Fulfilled
Version creation and switching	Create a version while retaining the analysis file, create another with a replacement file, choose all, none, or selected supplementary files, switch versions, edit a version, and delete a version.	Each version was created and selectable. Metadata editing and deletion worked, and deleting the selected version caused fallback to another available version.	Fulfilled
Project download	Download the analysis file only, download all files, choose output format, and test ZIP and TAR.GZ packaging.	The selected project version was prepared and downloaded using the requested scope and archive type.	Fulfilled
Owner-facing sharing controls	Generate and copy a secure link, disable and enable it, regenerate it, publish the project, and change the project back to private.	The owner-facing management controls and corresponding feedback operated as expected.	Fulfilled
Recipient-facing sharing	Open the generated public and secure URLs as an unauthenticated recipient.	Both URLs led to a 404 page and did not render the shared project.	Not fulfilled
File viewing	Attempt to open uploaded primary and supplementary files from Project Details and Files Overview.	The actions were present, but the uploaded files failed to open correctly.	Partially fulfilled

The positive conversion and missing-file outcomes are shown in Figure 6.1. The recipient-facing failure is shown separately at a larger size in Figure 6.2, so that the not-found result remains legible in the printed thesis.

The validation shows that the central workflows can generally be completed while the user remains within the implemented visitor or authenticated-owner interfaces. The main continuity failures occur when a recipient attempts to access a project through a generated public or secure URL and when a user attempts to view an individual uploaded file directly in the browser.

6.5 Evaluation Limitations

The results in this chapter must be interpreted within several substantial methodological and technical limitations. These limitations restrict the strength, reproducibility, and



(a) Successful guest conversion and download (b) Completed missing-file upload

Figure 6.1: Positive outcomes from the manual end-to-end validation

404 | This page could not be found.

Figure 6.2: Recipient-facing public or secure project route returning a not-found page

generalizability of the evaluation and prevent the results from being treated as evidence of production readiness.

First, no formal usability study was conducted. The feedback participants were QDArchive project supervisors and developers who already understood the platform goals, terminology, and project history. Their feedback was useful for identifying defects, questioning design decisions, and improving workflow clarity, but it cannot demonstrate that the interface is understandable or efficient for qualitative researchers, reviewers, repository staff, or other independent users.

Second, the prototype was designed, implemented, and partly evaluated by the same developer. This provides detailed implementation knowledge but introduces confirmation bias and the possibility that expected behavior influenced interpretation of the results. The evaluation attempts to reduce this risk by reporting observable outcomes, separating mechanisms from working behavior, documenting unsuccessful workflows, and incorporating supervisor and developer feedback. It does not eliminate the underlying limitation.

Third, no automated tests were implemented. No unit-test, integration-test, browser-based end-to-end test suite, or coverage report is available. The evaluation therefore depends on manual execution and source inspection. This reduces reproducibility, limits systematic regression detection, and means that later code changes could invalidate the reported results without being detected automatically.

Fourth, no standalone manual test protocol was recorded at the time of execution. The scenarios consolidated in Table 6.6 reconstruct the principal tested workflows, but they do not provide the same reproducibility as dated test cases containing fixed inputs, detailed preconditions, expected outcomes, actual outcomes, and repeated execution records. Internal GitHub tickets and discussions were used during development but do not form a complete independent evaluation dataset.

Fifth, browser and device coverage was limited. Chrome was the primary browser. Desktop testing used Windows and macOS, while responsive inspection relied largely on Chrome device emulation. Firefox, Safari, Edge, assistive browsers, a broad range of physical devices, unusual display scaling, and extensive zoom conditions were not tested systematically.

Sixth, accessibility was not evaluated formally. No automated accessibility tool, screen reader, complete keyboard-only walkthrough, or systematic contrast, focus-order, focus-trapping, and semantic-structure audit was used. The accessibility assessment is therefore based only on implementation inspection and selected interface behavior.

Seventh, conversion testing covered only selected `.qdpx`, `.refi`, and `.qda` inputs and targets exposed by the integrated conversion service. The evaluation did not execute a complete matrix of source-target combinations, very large files, every supported CAQDAS feature, deliberately malformed package structures, or all possible conversion-loss conditions. The correctness of the underlying conversion algorithms and corpus representation was also outside the frontend scope.

Eighth, several boundary and failure conditions were not exercised. These include expired password-reset links, prolonged session expiration, high project or file counts, concurrent metadata updates, simultaneous uploads, slow networks, intermittent connectivity, backend outages, storage failures, long-running conversions, and multiple concurrent conversion requests.

Ninth, no systematic performance evaluation was conducted. Page-load times, request latency, rendering performance, memory use, conversion duration, archive-generation time, large-project behavior, and performance under concurrent users were not measured.

Tenth, security was not independently evaluated. The assessment does not include penetration testing, authorization-bypass attempts, token-security review, secure-link entropy analysis, dependency vulnerability scanning, content-security-policy review, or systematic testing of malicious file uploads. The presence of authentication and access controls in the interface cannot be interpreted as independent evidence of system security.

Eleventh, privacy and data-protection behavior were not audited formally. The evaluation did not verify data-retention policies, erasure behavior, backup removal, access logging, consent compliance, or whether all public and secure visibility transitions were enforced correctly at every backend layer.

Twelfth, the deployed system was a development environment rather than a production installation. The evaluation does not cover production deployment, availability, monitoring, backups, disaster recovery, long-term storage, operational support, preservation guarantees, or service-level reliability.

Finally, the recorded supervisor videos, weekly discussions, and internal GitHub tickets are project materials rather than an independently curated evaluation dataset. They informed the iterative implementation but cannot support statistical conclusions or independent replication of the full feedback process.

6.6 Overall Assessment

The evaluation indicates that the QDArchive frontend prototype provides a substantial implementation of the main visitor and registered-researcher workflows. In particular, conversion, guided project creation, metadata editing, missing-file completion, supplementary-file management, versioning, project download, and owner-facing visibility and secure-link controls were implemented and manually exercised.

The prototype also demonstrates that QDA projects can be represented through a structured frontend rather than as a single undifferentiated file. The distinction between analysis files, referenced primary files, supplementary files, versions, missing references, metadata, and visibility states makes important aspects of the archival structure visible to the user.

The raw JSON view provides evidence that structured artifact data can be exposed for technical inspection. It should not, however, be interpreted as a completed domain-specific viewer. The prototype does not yet provide independent visual exploration of codes, coded segments, memos, annotations, or analytical relationships.

The strongest non-functional evidence concerns low entry barrier, transparency, interaction consistency, feedback and error handling, interoperability communication, and maintainable code organization. Public *Find* and *Convert* access reduces registration friction, while shared components, the theme system, service modules, query hooks, typed models, and contexts provide a coherent implementation structure.

Important functional discontinuities remain. Public and secure project links cannot be consumed by recipients because the corresponding routes return a not-found page. Uploaded-file viewing remains defective. Advanced archive discovery, complete publication, open-licence selection, named-user sharing, starred projects, notifications, data export, account

deletion, reviewer interfaces, administrator interfaces, and domain-specific artifact inspection remain unavailable.

The presentation also contains smaller defects, including the brief display of a project identifier in breadcrumbs and poor wrapping or overflow of long metadata values in some constrained layouts. These defects do not prevent the principal owner workflows from operating, but they reduce visual polish and should be corrected before broader deployment.

The evaluation cannot establish general usability, accessibility compliance, security, performance, production reliability, or long-term preservation. The absence of representative-user testing, automated regression testing, formal accessibility assessment, comprehensive browser and device coverage, security testing, and performance measurement prevents the prototype from being considered production-ready.

Overall, the evaluated frontend should be understood as a functional research prototype and an implementation foundation. It demonstrates that QDArchive's principal conversion and project-management capabilities can be presented through a coherent web interface, while also exposing the specific workflow, inspection, access, and quality-assurance work required for the platform to become an operational archival service. The following chapter translates these remaining gaps into prioritized development steps.

7 Next Steps

The evaluation in Chapter 6 showed that the QDArchive frontend provides a functional foundation for the principal visitor and registered-researcher workflows. Conversion, guided project creation, metadata and file management, missing-file completion, versioning, downloading, and owner-facing visibility and secure-link controls were implemented and manually validated.

The most important remaining work concerns workflows that are already exposed but cannot yet be completed. Public and secure links do not render recipient-facing project views, uploaded files cannot be viewed reliably, and public visibility does not yet provide a complete licence-aware publication process. These gaps should be addressed before broader extensions such as advanced discovery, collaboration, or personalization.

The recommended sequence is summarized in Table 7.1.

Order	**Next step**	**Related requirements**	**Rationale**
1	Recipient-facing public and secure access	FR-PM23, FR-PM25, FR-DA1	Generated URLs currently lead to a not-found page and interrupt the sharing workflow.
2	Reliable file viewing	FR-PM19	View actions exist, but uploaded primary and supplementary files do not open reliably.
3	Licence-aware publication and public inspection	FR-F9, FR-PM25, FR-DA1, FR-DA3	Licence selection, publication validation, and a complete public project view are missing.
4	Automated regression testing	NFR-12	The absence of automated tests increases the risk of regressions.
5	Accessibility validation and interface corrections	NFR-7	No systematic keyboard, screen-reader, contrast, or automated accessibility evaluation was performed.

Table 7.1: Recommended sequence for QDArchive frontend development

7.1 Immediate Functional Completion

The highest-priority task is to complete recipient-facing public and secure access. The current prototype allows owners to change visibility and manage secure links, but the resulting URLs do not render the intended recipient view.

Public and secure routes should provide read-only project views based on suitable parts of the authenticated project-detail interface. Recipients should be able to identify the project and version, inspect metadata and files, understand the applicable licence or access conditions, and determine whether downloads are permitted. Owner-specific controls for editing, deleting, versioning, and sharing should be excluded.

Secure-link routes must validate the supplied token, and disabled or regenerated links should no longer provide access. The interface should distinguish invalid links, deleted

projects, insufficient permissions, and temporary server errors. Private projects accessed through secure links must remain excluded from archive search.

File viewing should be completed as part of the same work. The frontend should verify access and select a suitable viewer for text, PDF, image, audio, and video files. Unsupported formats should provide an explanation and a download alternative. Missing files, unavailable storage objects, unsupported previews, and authorization failures should be communicated separately.

Publication should remain distinct from merely setting a project to public. A complete workflow should include:

- selection of an open licence;
- validation of mandatory metadata;
- confirmation of publicly accessible files;
- communication of reuse and download conditions; and
- a working public project-detail page.

The public project page should present available metadata, the selected or latest version, file roles, completeness state, licence, and download options. Selecting a result from *Find* should open this page. The raw JSON representation should remain a technical aid rather than the primary public artifact view. Exploration of codes, coded segments, memos, and analytical relationships can be added incrementally.

After public access is complete, *Find* can be extended with filters for tags, file format, licence, author, and dates. Pagination should preserve the active query and filters. File-level search, richer result cards, saved searches, and recommendations should remain lower priorities.

7.2 Quality Assurance

Automated testing is the main technical quality improvement required after the incomplete workflows are finished. Unit tests should cover format helpers, metadata validation, query-key construction, formatting utilities, and request construction. Component and integration tests should cover shared controls, dialogs, authentication-dependent rendering, queries, mutations, and cache invalidation.

Browser-based end-to-end tests should cover authentication, guest conversion, project creation, missing-file completion, file management, versioning, downloading, visibility and secure-link management, recipient access, and file viewing. Negative scenarios should include unsupported files, invalid credentials, failed conversion, unavailable projects, disabled links, network failures, and incomplete publication metadata. The scenarios in Table 6.6 can provide the basis for this suite.

Accessibility should be evaluated systematically against WCAG 2.2 (World Wide Web Consortium, 2024). Testing should include keyboard navigation, focus order, dialogs and step-based forms, screen-reader output, colour contrast, zoom behavior, icon-only controls, and alternatives to drag-and-drop upload. Automated tools such as Lighthouse, axe, or WAVE should be complemented by manual testing.

Smaller interface defects should be corrected during the same phase. Breadcrumbs should not expose internal identifiers while loading, long metadata should wrap or expand acces-

sibly, and placeholder functionality should consistently display an explicit *Not available yet* state.

7.3 Further Extensions

After the access, publication, testing, and accessibility gaps have been addressed, the frontend can be extended to additional roles and account workflows.

Named-user sharing should allow owners to invite registered users, review access, and revoke permissions. Shared projects could use the same read-only presentation developed for secure-link recipients. Reviewer interfaces could build on this model, while comments, response histories, and requests for additional files remain later extensions.

Administrator functionality may include review queues, project-status management, moderation, ingestion errors, and user administration. High-impact actions should require confirmation and eventually be recorded through an audit mechanism. Account-data export, account deletion, and notification preferences should also be completed.

Longer-term possibilities include multiple editors, richer collaboration, internationalization, alternative themes, external repository integration, saved searches, personal collections, and metadata-based recommendations. The reusable component set could also develop into a documented QDArchive design system.

The immediate objective is therefore to turn the current owner-oriented prototype into a continuous end-to-end system in which projects can be shared or published, opened by recipients, inspected, and downloaded under clearly communicated access conditions. Automated testing and accessibility validation would then provide the foundation for further expansion.

8 Conclusion

This thesis investigated, designed, implemented, and evaluated the first frontend prototype of QDArchive, a web-based platform for managing, converting, finding, accessing, and sharing qualitative data analysis artifacts.

QDA artifacts combine primary research material with analytical structures such as codes, coded segments, memos, annotations, and relationships. Their frontend representation therefore requires more than conventional file upload and download. Users must be able to understand metadata, file roles, missing references, versions, formats, visibility states, and available actions without detailed knowledge of the underlying infrastructure.

Using a requirements-driven and iterative development process, this thesis produced and evaluated a functional prototype for the principal visitor and registered-researcher workflows.

8.1 Summary and Contributions

The related-work analysis showed that the repositories considered in this thesis support storage, metadata, access management, and reuse, but generally treat CAQDAS projects as files rather than structured analytical objects. Based on this problem context and the QDArchive project documentation, the thesis established a frontend requirement baseline organized around *Find*, *Convert*, and *Share*.

The first contribution is a frontend-specific requirements model connecting domain concerns such as interoperability, file roles, missing references, versioning, and controlled access with user roles, journeys, functional requirements, and quality expectations.

The second contribution is the frontend design, including the information architecture, route structure, interaction model, responsive layout strategy, and visual identity. Public workflows are separated from authenticated project management, while complex operations are presented through guided steps, dialogs, structured file sections, and explicit system states.

The principal contribution is the implemented prototype. It supports public onboarding and archive search, guest and authenticated conversion, authentication, a researcher dashboard, guided project creation, metadata and file management, missing-file completion, versioning, configurable downloads, profile settings, and owner-facing visibility and secure-link controls.

The implementation also provides a maintainable structure by separating routes, feature and shared components, query hooks, services, contexts, typed models, utilities, and theme configuration. Interface, server, authentication, browser-local, and global feedback state are managed according to their scope.

Finally, the evaluation distinguishes visible controls from completed workflows. Owner-facing sharing was evaluated separately from recipient access, file-view controls from successful file viewing, and raw JSON output from a domain-specific artifact viewer.

8.2 Answer to the Research Question and Limitations

The central research question asked:

How can a frontend prototype for QDArchive support the user-facing workflows required for archiving, managing, finding, and accessing QDA artifacts?

The results indicate that these workflows can be supported by separating low-barrier public functionality from authenticated project management, representing projects through explicit metadata, file roles, versions, completeness states, and access conditions, and guiding users through structured and state-aware interactions.

Visitors can search public projects and convert supported files without registration. Registered researchers can create and manage projects, provide metadata, complete missing primary files, add supplementary material, create versions, configure downloads, and manage owner-facing visibility and secure links. Together, these functions form a connected workflow from conversion and project creation to editing, versioning, downloading, and preparation for sharing.

The contribution of this thesis is limited to the frontend and its integration with available services. It defines how users navigate the application, provide input, initiate operations, interpret project states, and receive feedback. It does not implement or guarantee backend authorization, storage, backup and recovery, long-term preservation, or the semantic correctness of format conversion. QDConvert, backend services, and storage remain separate system responsibilities.

Two principal functional limitations remain. First, public and secure links do not yet render recipient-facing project views. Second, file and artifact inspection is incomplete: uploaded files do not open reliably, and the prototype does not yet provide domain-specific exploration of codes, coded segments, memos, annotations, or relationships.

The evaluation was based on manual scenario validation, implementation inspection, and formative feedback from supervisors and developers. No formal representative-user study, automated regression suite, comprehensive accessibility audit, security assessment, performance evaluation, or broad cross-browser evaluation was conducted. The findings therefore demonstrate prototype feasibility in the tested environment rather than production readiness or general user acceptance.

8.3 Overall Conclusion

This thesis achieved its objective of establishing a frontend requirement baseline, translating the QDArchive domain into a user-facing design, implementing the principal visitor and researcher workflows, and evaluating the resulting prototype.

The work demonstrates that QDA projects can be represented through a structured web interface that communicates metadata, file roles, missing references, versions, formats, download options, and access states. It also shows that conversion, project creation, file completion, project management, versioning, downloading, and owner-facing access configuration can be connected within one application.

The result is not a completed archival service but a functional and evaluated frontend foundation. Completing recipient-facing access, reliable file viewing, licence-aware publication, public artifact inspection, automated testing, and accessibility validation remains the main path toward a more complete QDArchive platform.

References

Antes, A. L., Walsh, H. A., Strait, M., Hudson-Vitale, C. R., & DuBois, J. M. (2018). Examining data repository guidelines for qualitative data sharing. *Journal of Empirical Research on Human Research Ethics*, *13*(1), 61–73.

arXiv. (2024). *Arxiv.org e-print archive*. https://arxiv.org

Bishop, L. (2009). Ethical sharing and reuse of qualitative data. *Australian Journal of Social Issues*, *44*(3), 255–272.

Borgman, C. L. (2012). The conundrum of sharing research data. *Journal of the American Society for Information Science and Technology*, *63*(6), 1059–1078.

Cliggett, L. (2013). Qualitative data archiving in the digital age: Strategies for data preservation and sharing. *The Qualitative Report*, *18*(24), 1.

Corti, L. (2007). Re-using archived qualitative data–where, how, why? *Archival Science*, *7*(1), 37–54.

Corti, L., & Gregory, A. (2011). Caqdas comparability. what about caqdas data exchange? *Forum: Qualitative social research*, *12*(1).

Creswell, J. W., & Poth, C. N. (2018). Qualitative inquiry and research design: Choosing among five approaches 4/e. *Sage*.

Crosas, M. (2011). The dataverse network®: An open-source application for sharing, discovering and preserving data. *D-lib Magazine*, *17*(1/2).

Denzin, N. K., & Lincoln, Y. S. (2011). *The sage handbook of qualitative research*. Sage publications.

Di Gregorio, S., & Davidson, J. (2009). *Qualitative research design for software users*. McGraw-Hill Education (UK).

Doorn, P. (2018). Science europe practical guide to the international alignment of research data management. *KNAW Research Portal*.

European Organization for Nuclear Research (CERN) & OpenAIRE. (2013). *Zenodo*. https://zenodo.org

Evers, J., Caprioli, M. U., Nöst, S., & Wiedemann, G. (2020). What is the refi-qda standard: Experimenting with the transfer of analyzed research projects between qda software. *Forum Qualitative Sozialforschung/Forum: Qualitative Social Research*, *21*(2), 37.

Figshare. (2024). *Figshare*. https://figshare.com

Foster, E. D., & Deardorff, A. (2017). Open science framework (osf). *Journal of the Medical Library Association: JMLA*, *105*(2), 203.

Hocker, J., Bipat, T., McDonald, D. W., & Zachry, M. (2021). Evaluating qualico: An ontology to facilitate qualitative methods sharing to support open science. *Journal of Internet Services and Applications*, *12*(1), 5.

ISO/IEC/IEEE. (2017). *Systems and software engineering — vocabulary*. International Organization for Standardization.

Jackson, K., & Bazeley, P. (2019). Qualitative data analysis with nvivo. *Sage*.

Karcher, S., Kirilova, D., & Weber, N. (2016). Beyond the matrix: Repository services for qualitative data. *IFLA journal*, *42*(4), 292–302.

Miles, M. B., Huberman, A. M., & Saldana, J. (2014). *Qualitative data analysis*. sage.

Muhr, T. (2000). Increasing the reusability of qualitative data with xml. *Forum: Qualitative Social Research*, *1*(3).

MUI. (2026). *Theming*. https://mui.com/material-ui/customization/theming/

OECD. (2021). Recommendation of the council concerning access to research data from public funding. https://legalinstruments.oecd.org/en/instruments/OECD-LEGAL-0347

Paulus, T., Lester, J., & Dempster, P. (2013). *Digital tools for qualitative research*. Sage.

Qualitative Data Repository. (2024). *Qualitative data repository*. Syracuse University. https://qdr.syr.edu

Saldaña, J. (2021). The coding manual for qualitative researchers. *Sage*.

Silver, C., & Lewins, A. (2014). Using software in qualitative research: A step-by-step guide. *Sage*.

TanStack. (2026). *Query keys*. https://tanstack.com/query/v5/docs/framework/react/guides/query-keys

UK Data Service. (2024). *Uk data service*. https://ukdataservice.ac.uk

Vercel. (2026). *File-system conventions: Route groups*. https://nextjs.org/docs/app/api-reference/file-conventions/route-groups

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., et al. (2016). The fair guiding principles for scientific data management and stewardship. *Scientific data*, *3*(1), 1–9.

Wolski, U. (2018). The history of the development and propagation of qda software. *The Qualitative Report*, *23*(13), 6–20.

World Wide Web Consortium. (2024, December 12). *Web content accessibility guidelines (wcag) 2.2*. https://www.w3.org/TR/WCAG22/

Appendix A Internal QDArchive Project Documents

The requirements for the frontend prototype were derived in part from two internal QDArchive project documents. These documents were provided as project material and were used to understand the broader system vision, the intended user-facing workflows, and the scope of the frontend prototype. The documents are internal project materials and are therefore not reproduced in full in this thesis. They may be made available to examiners upon request, subject to the approval of the QDArchive project supervisors.

Their role in the requirements derivation is summarized in Table A.1.

Internal document	**Use in this thesis**
Original QDArchive project description	Used to understand the broader goal of QDArchive as a web-based service for archiving, providing, reviewing, and converting QDA artifacts. It informed the general scope of the platform, including artifact management, access, review, download, and interoperability-related requirements.
Updated QDArchive PRD	Used to derive the main user-facing solution areas of the frontend: *Find*, *Convert*, and *Share*. It informed the organization of the frontend requirements, user journeys, and workflow-oriented design decisions.

Table A.1: Internal QDArchive project documents used during requirements derivation

Appendix B Frontend Screenshot Gallery

This appendix provides additional screenshots of the implemented QDArchive frontend prototype. The screenshots complement the representative figures shown in Chapter 4 and document the main public and authenticated workflows of the application.

B.1 Public Entry and Authentication

Figure B.1: QDArchive landing page: Hero

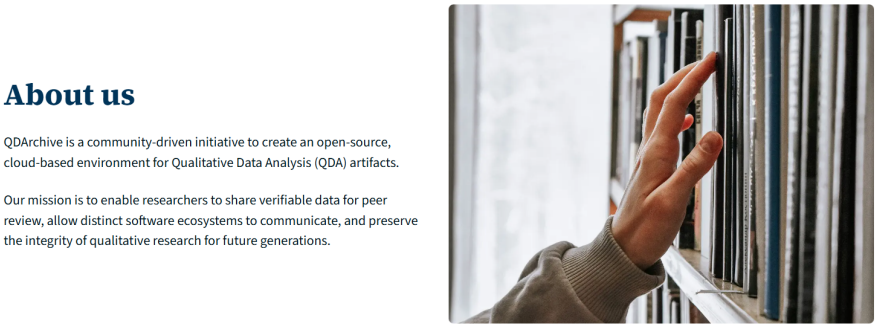


Figure B.2: QDArchive landing page: About us

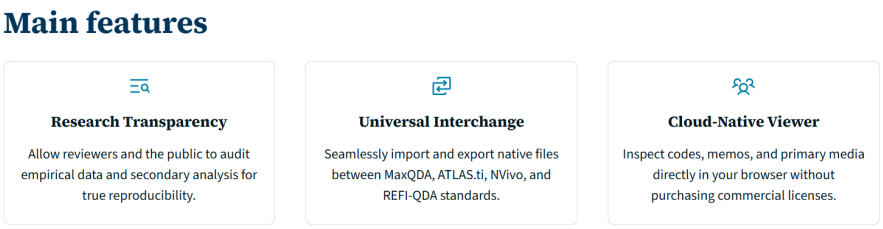


Figure B.3: QDArchive landing page: Main features

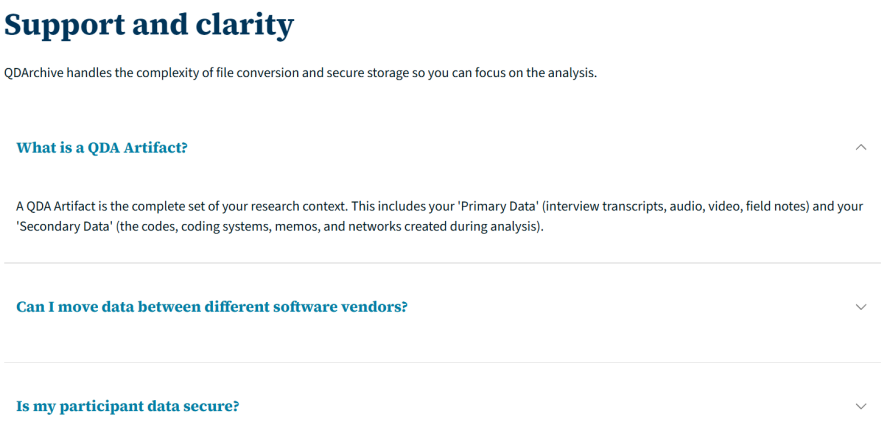


Figure B.4: QDArchive landing page: FAQ

Figure B.5: QDArchive landing page: Footer

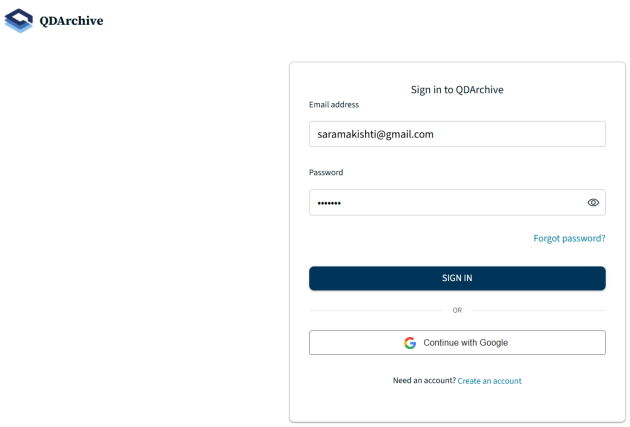


Figure B.6: Sign-in view

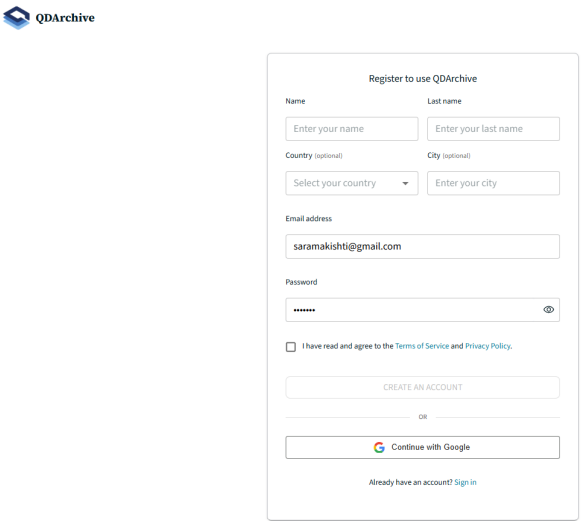


Figure B.7: Registration view

B.2 Find Workflow

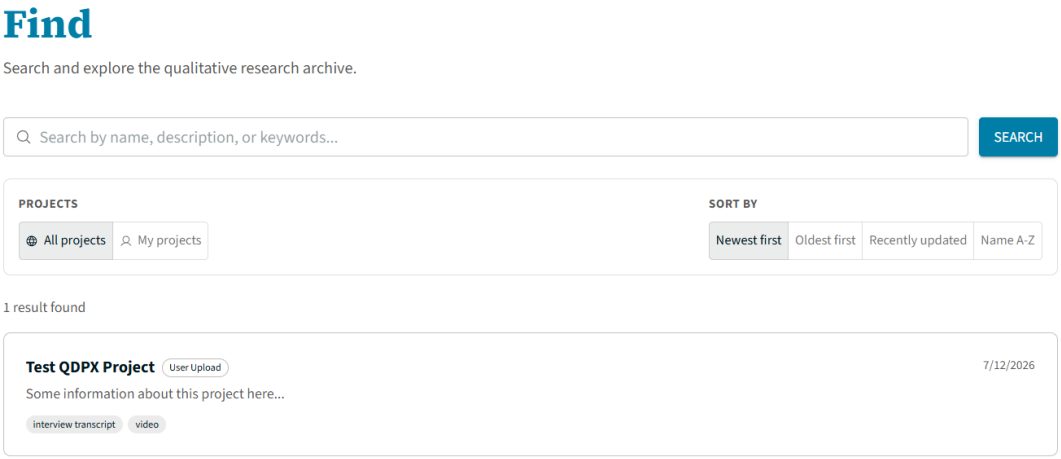


Figure B.8: Find workflow with public project results

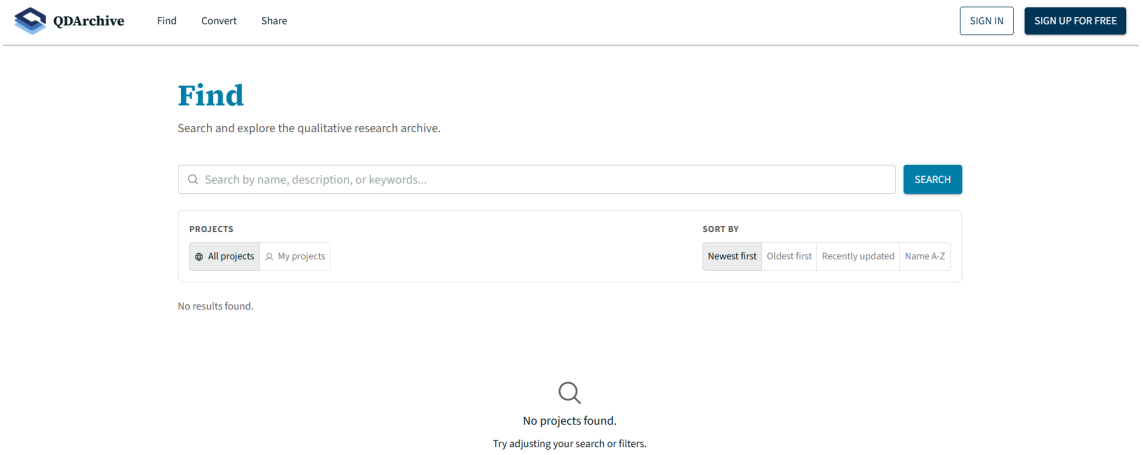


Figure B.9: Find workflow empty state

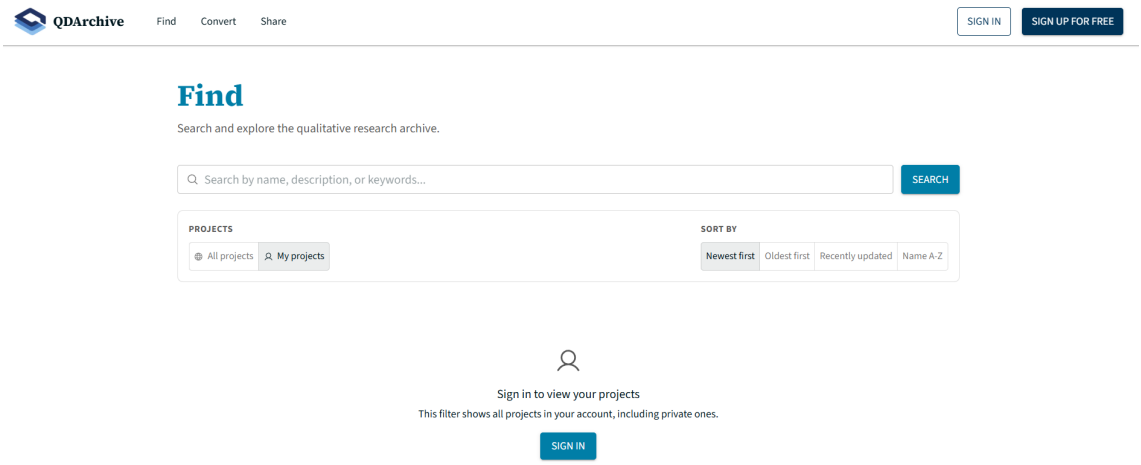


Figure B.10: Find workflow for unauthenticated users

B.3 Convert Workflow

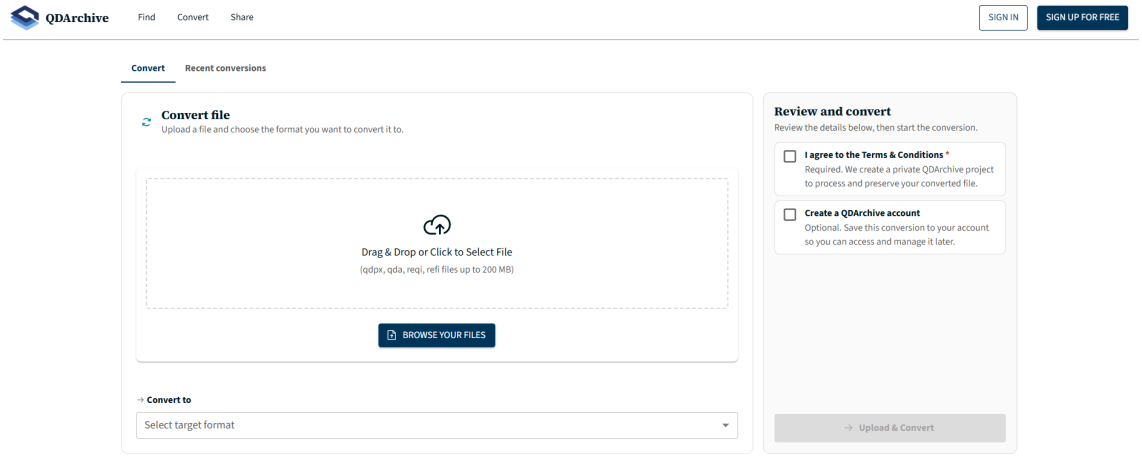


Figure B.11: Convert workflow with file upload

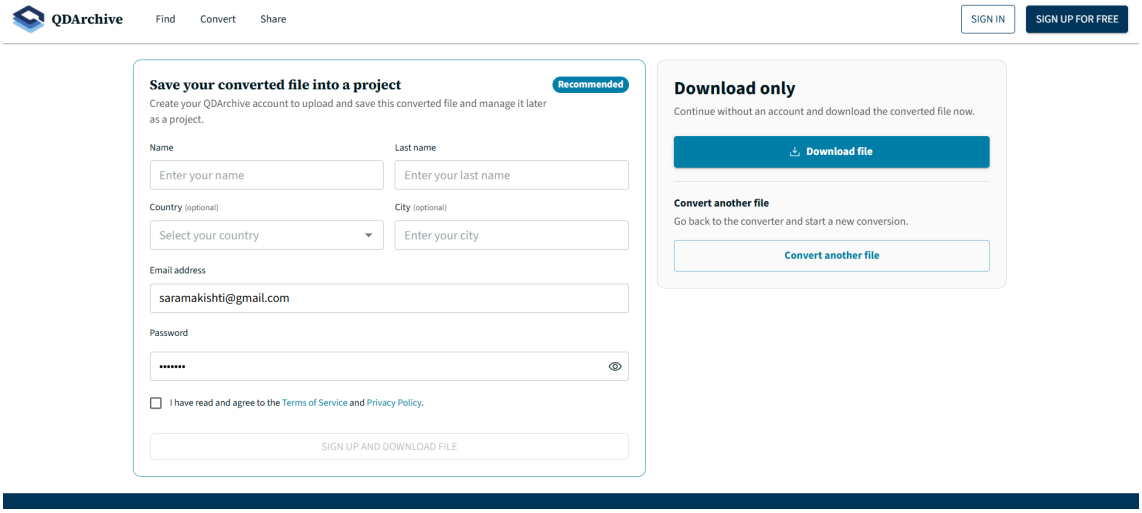


Figure B.12: Convert workflow after successful conversion with sign-up chosen

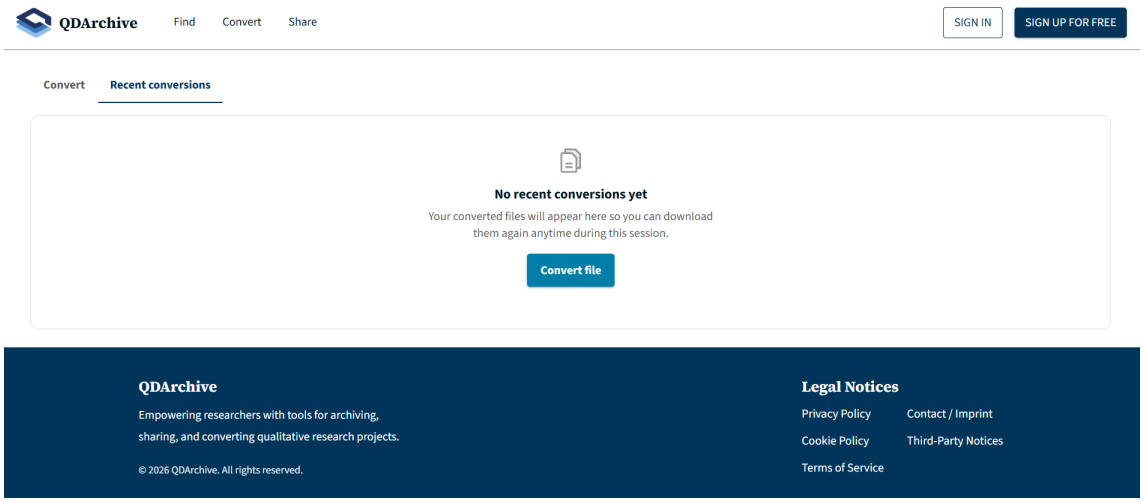


Figure B.13: Recent conversions view

B.4 Dashboard and User Settings

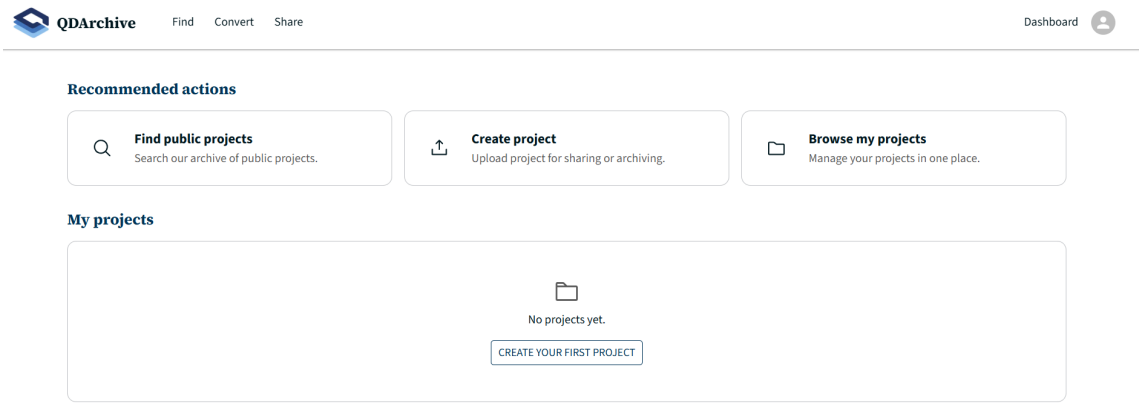


Figure B.14: Authenticated dashboard: Top part

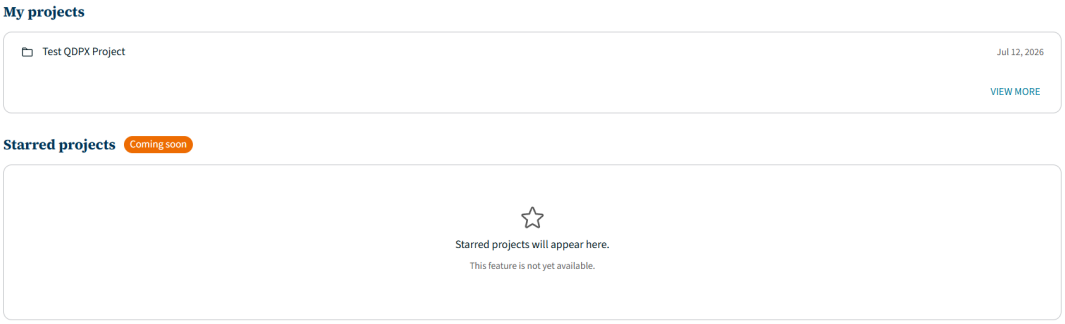


Figure B.15: Authenticated dashboard: Bottom part

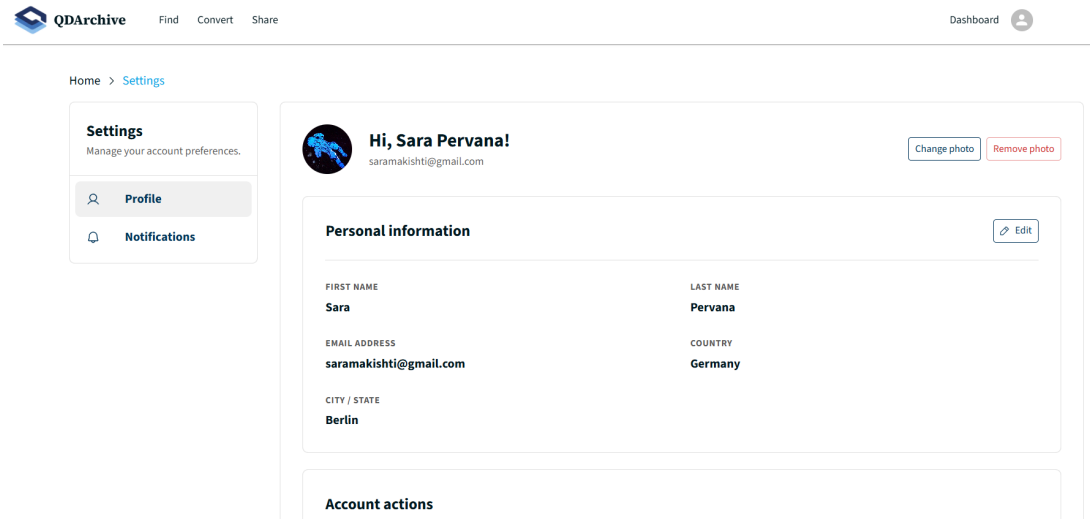


Figure B.16: Profile settings view

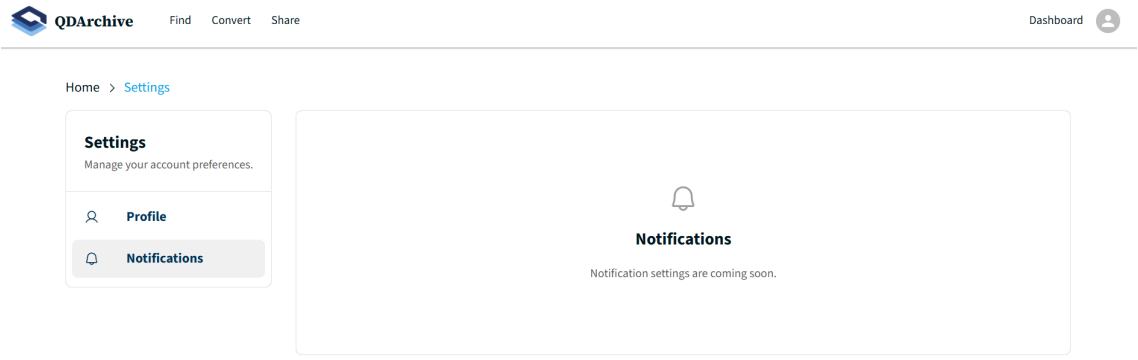


Figure B.17: Notification settings placeholder

B.5 Project Creation

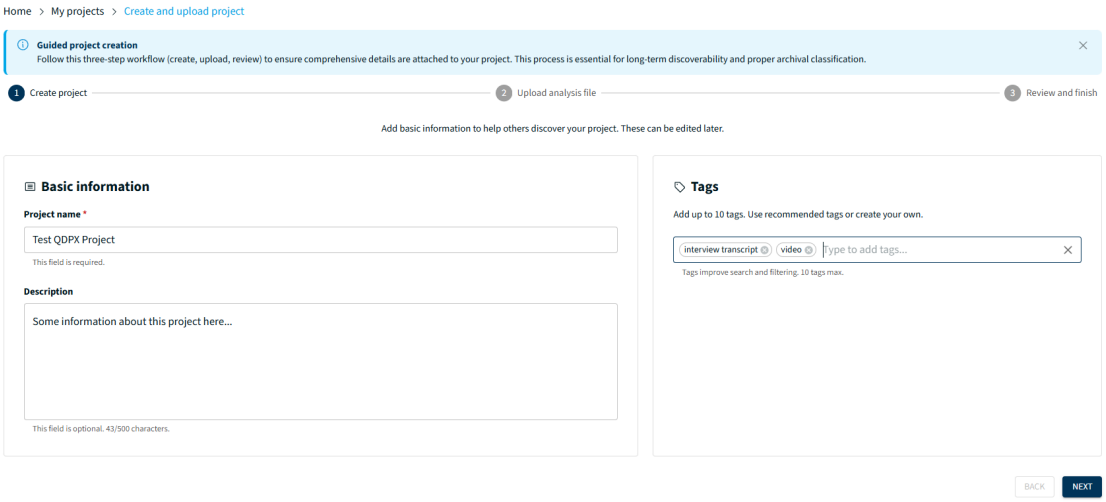


Figure B.18: Project creation: metadata step

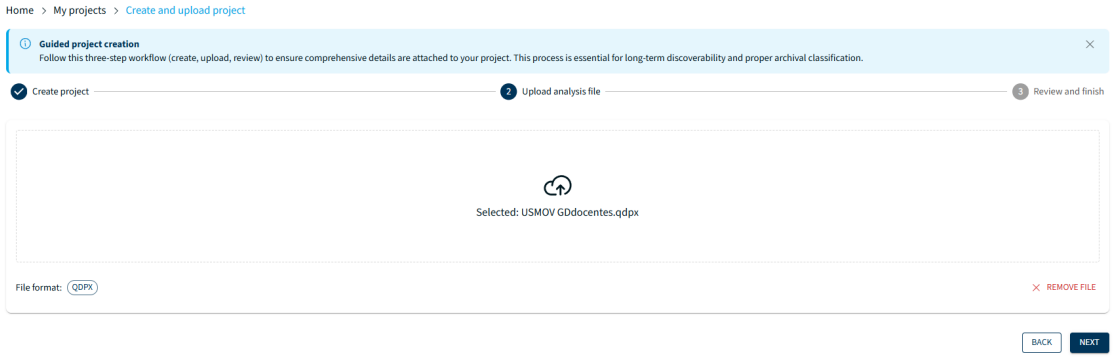


Figure B.19: Project creation: analysis file upload step

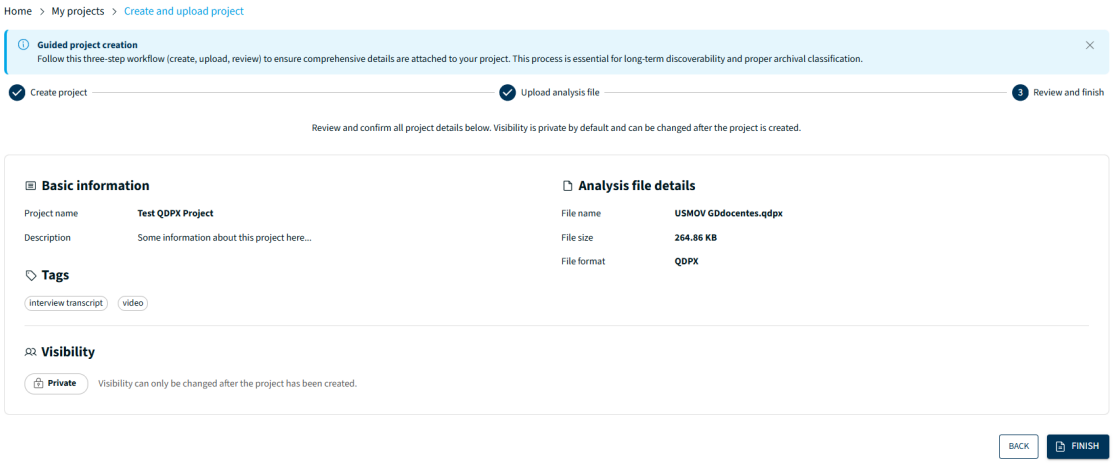


Figure B.20: Project creation: review step

B.6 Project Details and File Management

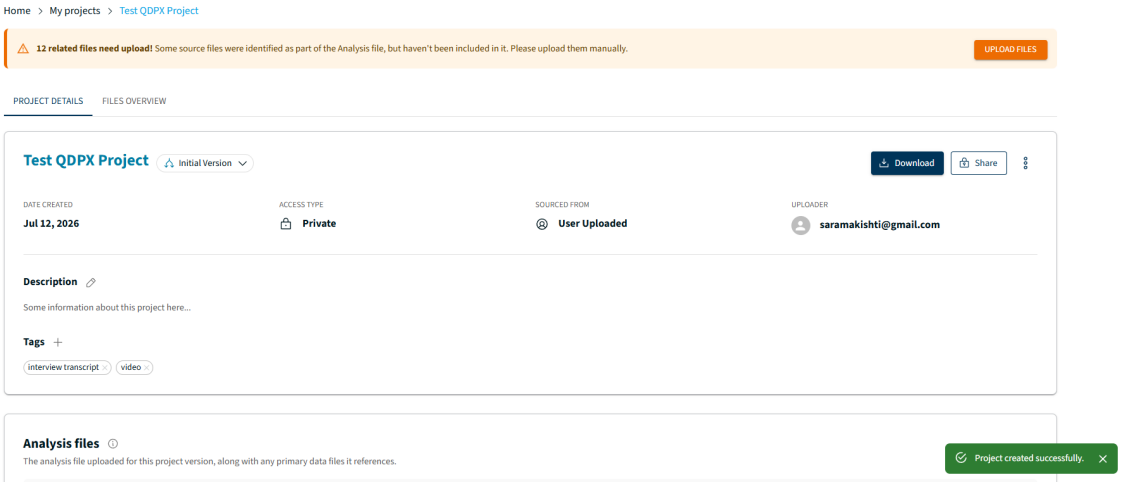


Figure B.21: Project detail view with metadata and file sections

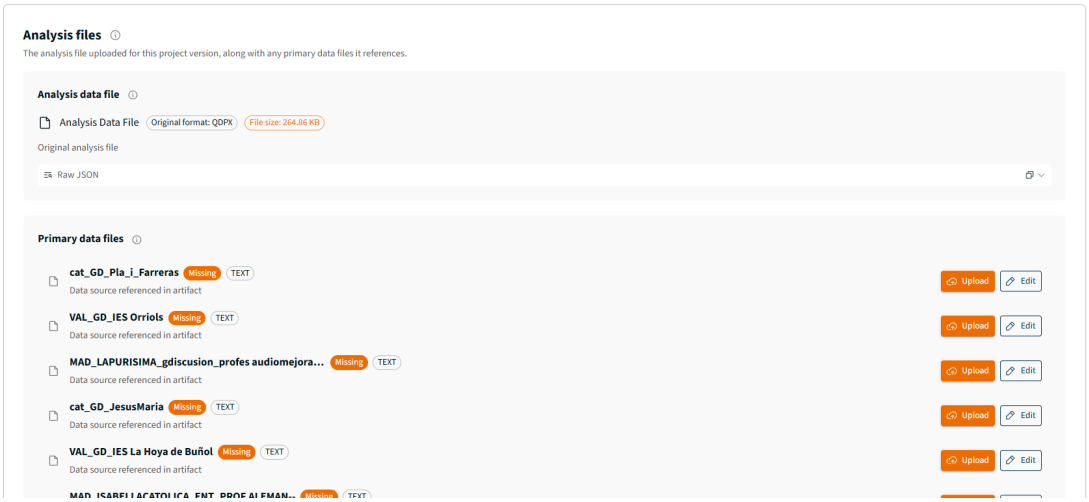


Figure B.22: Project detail view with missing referenced files

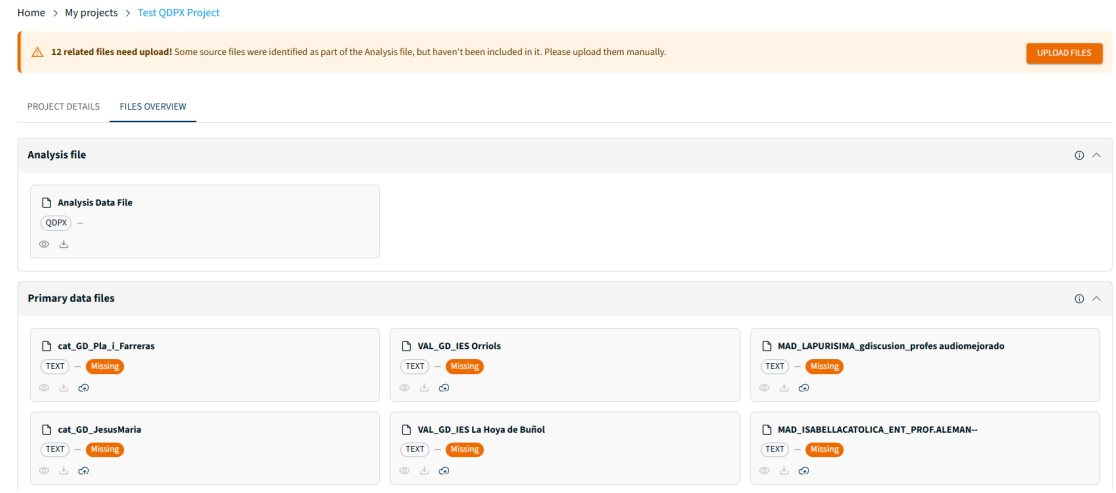


Figure B.23: Files overview tab

B.7 Download, Versioning, and Sharing

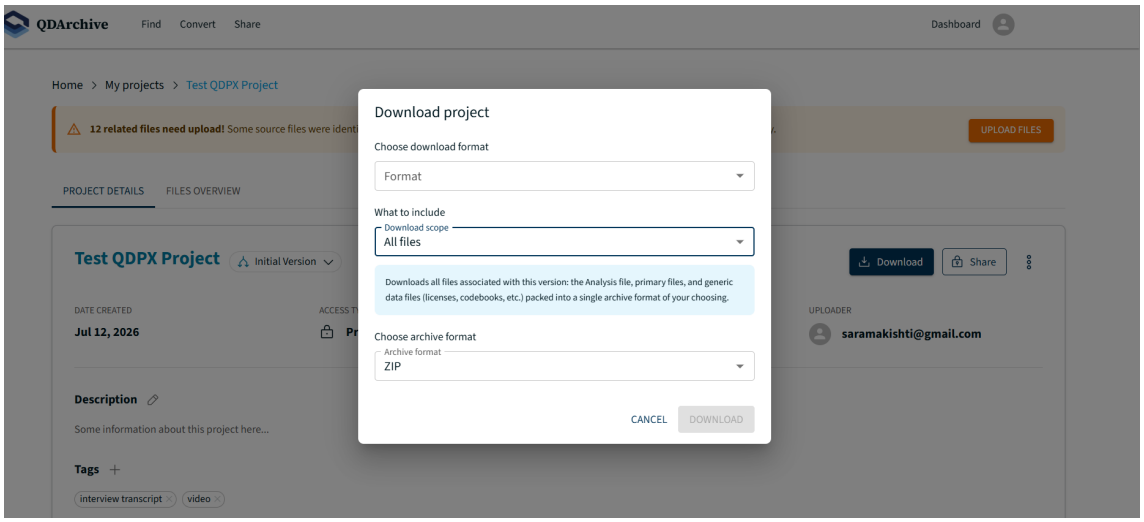


Figure B.24: Project download options

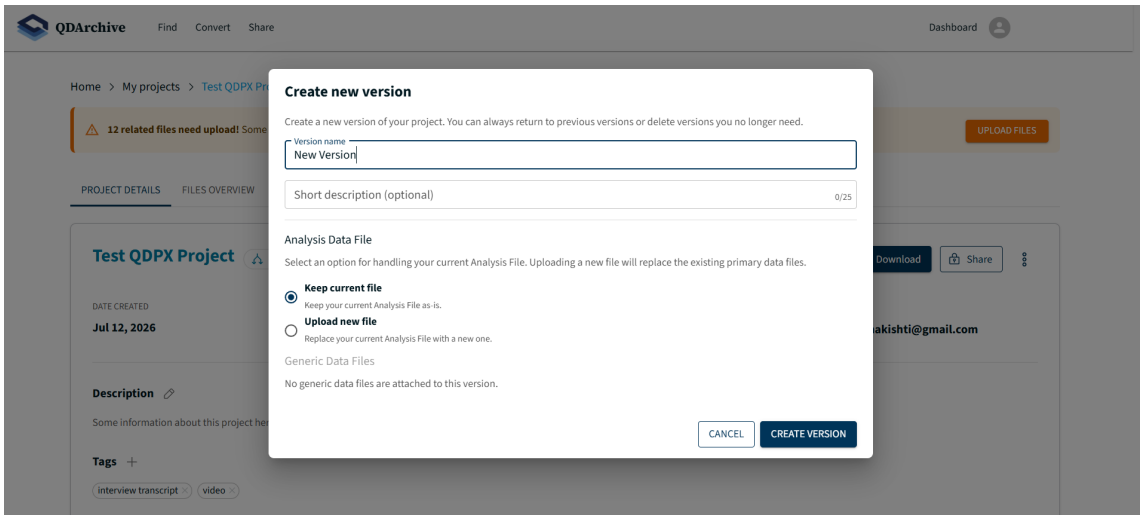


Figure B.25: Project versioning workflow: Keep current file

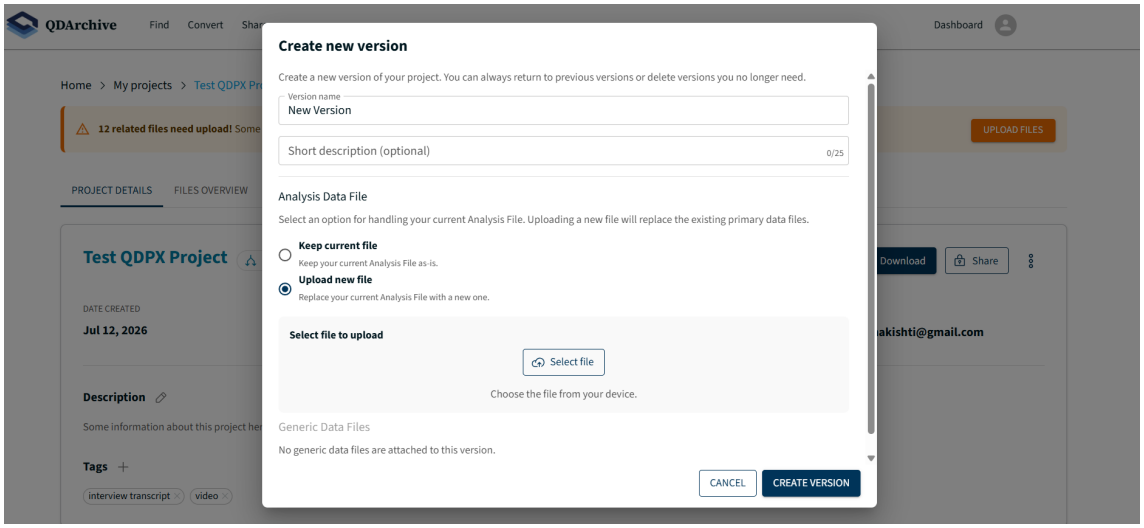


Figure B.26: Project versioning workflow: Upload new file

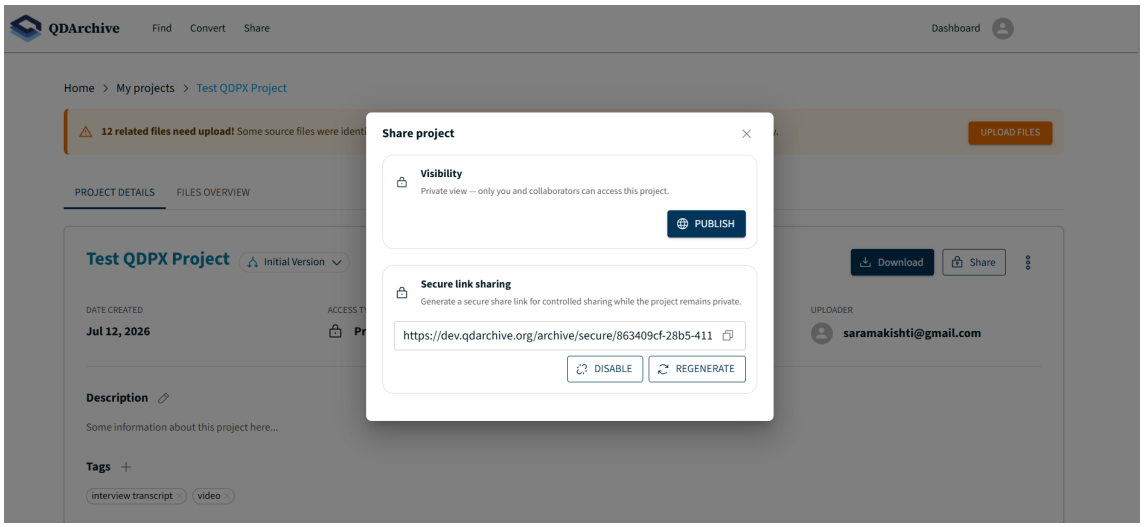


Figure B.27: Project sharing and secure link controls: Publish project

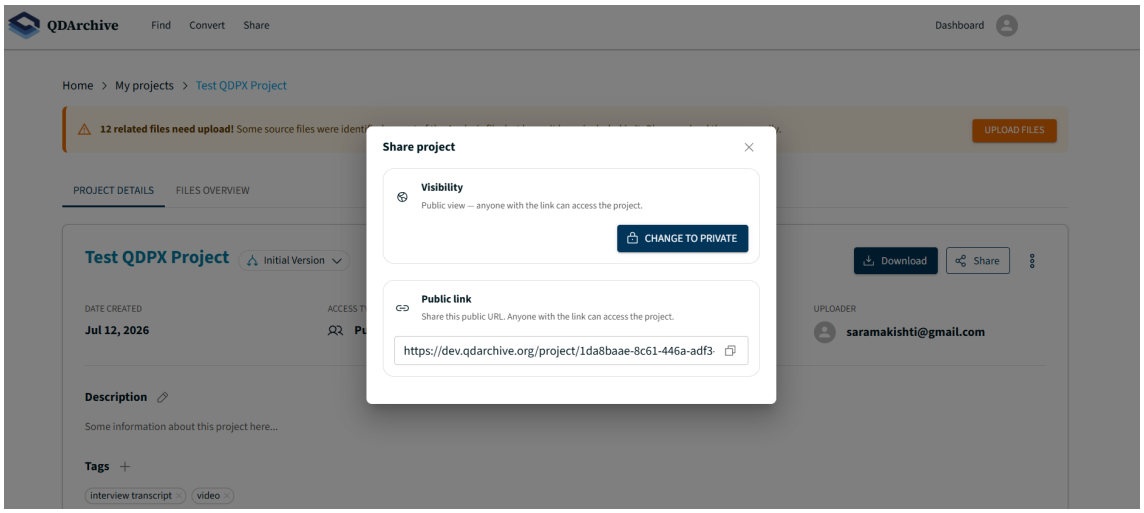


Figure B.28: Project sharing and secure link controls: Change to private