QDA Austauschformate in QDAcity

MASTER THESIS

Wingkin Mak

Eingereicht am 2. Januar 2025

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik
Professur für Open Source Software

Betreuer:
Andreas Kaufmann
Prof. Dr. Dirk Riehle, M.B.A.

Eidesstattliche Erklärung

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 2. Januar 2025

Lizenz

Erlangen, 2. Januar 2025

Anerkennnug

Ich erkenne die Hilfe von ChatGPT bei der Verbesserung des Schreibens und ausschließlich des Schreibens dieses Artikels an.

Abbildungen wurden dank PlantText-UML-Editor[1] generiert.

[1]https://www.planttext.com/

Abstract

There are many QDA software solutions available on the market, and QDAcity is one of them. Each of these solutions has its own advantages and disadvantages. It can happen that researchers switch from one QDA software application to another, whether voluntarily or not. Before the start of this master's thesis, it was not possible to export data from QDAcity to use it in other QDA software. This limitation reduced the attractiveness of QDAcity, which could result in it being used less or not chosen at all. The lack of export functionality also made it difficult for researchers to archive their projects or share their results with others.

In this work, the ability to exchange data between QDAcity and other QDA software solutions was improved. To this end, the REFI-QDA standard was introduced into QDAcity, enabling the import and export of codebooks and projects. This standard is used by many QDA software solutions, which significantly enhances the interoperability of QDAcity. The goal of this work was to explore how data in QDAcity is processed and exported into a format that can be imported into other QDA software. Additionally, new export formats were implemented to assist QDAcity users in further processing their analyses. Besides the export, the import functionality was also extended, allowing projects from other QDA software solutions to be imported into QDAcity. All these measures aim to make QDAcity more attractive from the researchers' perspective.

Zusammenfassung

Es gibt viele QDA-Softwarelösungen auf dem Markt, darunter auch QDAcity. Da jede Lösung sowohl Vor- als auch Nachteile bietet, kann es passieren, dass Forscher von einer QDA-Softwareanwendung zu einer anderen wechseln, entweder freiwillig oder gezwungenermaßen. Vor Beginn dieser Arbeit war es in QDAcity nicht möglich, Daten zu exportieren, um sie in anderen QDA-Softwareanwendungen zu nutzen. Dies stellte eine Einschränkung dar und könnte die Attraktivität von QDAcity verringern, was möglicherweise dazu führen könnte, dass die Software weniger verwendet oder gar nicht gewählt wird. Der fehlende Export erschwert es Forschern zudem, ihre Projekte oder Ergebnisse zu archivieren oder mit anderen zu teilen.

In dieser Arbeit wurde deshalb die Möglichkeit geschaffen, den Datenaustausch zwischen QDAcity und anderen QDA-Softwarelösungen zu verbessern. Dazu wurde der REFI-QDA-Standard in QDAcity eingeführt, der den Import und Export von Codebüchern und Projekten ermöglicht. Dieser Standard wird von vielen QDA-Softwarelösungen verwendet, was die Interoperabilität von QDAcity erheblich steigert. Ziel dieser Arbeit war es, zu untersuchen, wie Daten in QDAcity verarbeitet und in ein Format exportiert werden, das in anderen QDA-Softwarelösungen importiert werden kann. Zudem wurden neue Exportformate implementiert, die den Nutzern von QDAcity helfen, ihre Analysen weiterzuverarbeiten. Neben dem Export wurde auch der Import erweitert, sodass nun auch Projekte aus anderen QDA-Softwarelösungen in QDAcity integriert werden können. All diese Maßnahmen sollen QDAcity aus der Perspektive der Forschenden attraktiver machen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Akronyme

CES Collaborative Editing Service

GCS Google Cloud Storage

QDA qualitative Datenanalyse

1 Einleitung

Die qualitative Datenanalyse (QDA) ist ein zentraler Bestandteil vieler sozialwissenschaftlicher und interdisziplinärer Forschungsprojekte. Sie ermöglicht es, umfangreiche Text- oder andere unstrukturierte Daten systematisch zu untersuchen, um Muster, Bedeutungen und Zusammenhänge zu erkennen. Dabei kommt die Software QDAcity ins Spiel, die Forschenden hilft, diesen Prozess effizient zu gestalten.

Ein Forschungsprojekt in QDAcity startet typischerweise mit der Organisation der Rohdaten, wie Transkriptionen von Interviews, Beobachtungsprotokollen oder offenen Antworten aus Fragebögen. Dieses sogenannte Projekt dient als zentrale Einheit, in der alle Daten, Analysekategorien und Ergebnisse verwaltet werden. Im Projekt werden nicht nur die Datensätze gespeichert, sondern auch die später entwickelten Codes und die daraus resultierende Analyse.

Der Kern der qualitativen Analyse ist die sogenannte Codierung. Hierbei werden einzelne Textstellen aus den Daten mit Codes versehen, also Schlagwörtern oder Kategorien, die deren Inhalt oder Bedeutung zusammenfassen. Ein Beispiel: Ein Interviewausschnitt, in dem eine Person über ihre Arbeitszufriedenheit spricht, könnte mit dem Code „Arbeitszufriedenheit“ markiert werden. Dies ermöglicht es, ähnliche Aussagen später systematisch zu analysieren und zu vergleichen.

Damit dieser Prozess konsistent bleibt, wird ein Codebuch erstellt, das als strukturierte Sammlung aller Codes dient. Es definiert jeden Code und gibt Hinweise, wann und wie er anzuwenden ist. Das Codebuch spielt eine entscheidende Rolle, insbesondere wenn mehrere Forschende an einem Projekt arbeiten. Es sichert, dass alle Beteiligten einheitlich codieren und die Ergebnisse vergleichbar bleiben.

QDAcity unterstützt Forschende dabei, diese Arbeitsschritte effizient umzusetzen. Die Software bietet intuitive Werkzeuge zur Codierung, zur Verwaltung des Codebuchs und zur Erstellung eines übersichtlichen Analyseprojekts. Dadurch wird nicht nur der Arbeitsprozess erleichtert, sondern auch die Qualität und Nachvollziehbarkeit der Analyse erhöht.

1.1 Motivation

Zu Beginn dieser Arbeit bestand keine Möglichkeit, Projekte in QDAcity zu archivieren oder zu exportieren. Diese Funktionalität ist jedoch essenziell, beispielsweise wenn ein Nutzer im Rahmen einer wissenschaftlichen Arbeit das analysierte Projekt als Beleg einreichen muss. Ohne eine Exportfunktion war dies nicht umsetzbar.

Ein weiteres Problem bestand darin, dass Nutzer nicht zwischen verschiedenen QDA-Softwareanwendungen wechseln konnten, selbst wenn dies erforderlich war. Diese Einschränkung reduzierte die Attraktivität von QDAcity erheblich.

1.2 Ziel

Das Ziel dieser Arbeit ist es, die Interoperabilität von QDAcity durch die Erweiterung der Import- und Exportmöglichkeiten zu verbessern. Dabei wurden auch die Wünsche der aktiven Nutzer berücksichtigt. Besonders häufig wurde der Wunsch geäußert, Daten in einem spezifischen Format exportieren zu können, um die Arbeit in der qualitativen Datenanalyse zu erleichtern und eine Weiterverarbeitung der Daten zu ermöglichen.

1.3 Gliederung

Im folgenden Kapitel werden die Anforderungen vorgestellt, anhand derer bewertet werden soll, ob das Ziel erreicht wurde. Daraufhin wird die Architektur von QDAcity sowie die Art und Weise, wie die Daten in QDAcity gespeichert sind, erläutert. Im Anschluss werden die Designentscheidungen behandelt, insbesondere, welche Formate gewählt wurden und wie die Daten formatiert werden, um den Datenaustausch zu ermöglichen. Ein zentrales Thema hierbei ist der REFI-QDA-Standard, der von vielen QDA-Softwareanwendungen genutzt wird. Danach folgt die Umsetzung der neuen Import-/Export-Formate. Im abschließenden Kapitel wird bewertet, ob die Anforderungen erfüllt wurden, und ein Vergleich angestellt, inwiefern jetzt mehr Daten mit anderen QDA-Softwarelösungen ausgetauscht werden können.

2 Anforderungen

Im Folgenden werden die funktionalen und nicht-funktionalen Anforderungen aufgelistet. Sie wurden formuliert, um abschließend bewerten zu können, ob das geplante Ziel erreicht wurde.

2.1 Funktionale Anforderungen

2.1.1 Projekt importieren/exportieren

Ein Projekt in QDAcity umfasst die Dokumente, das Codebuch und die Codierungen. Ziel des Projekt-Imports und -Exports ist es einerseits, eine Archivierungsmöglichkeit für Projekte bereitzustellen, und andererseits, den Austausch von Projekten zwischen QDAcity und anderen QDA-Softwarelösungen zu ermöglichen. Daraus ergeben sich die folgenden funktionalen Anforderungen:

- **FA-01:** Der Nutzer kann ein Projekt aus QDAcity exportieren.
- **FA-02:** Der Nutzer kann ein Projekt in QDAcity importieren.
- **FA-03:** Der Nutzer kann ein Projekt aus QDAcity in eine andere QDA-Software exportieren.
- **FA-04:** Der Nutzer kann ein Projekt aus einer anderen QDA-Software in QDAcity importieren.
- **FA-05:** Der Nutzer kann das Projekt lokal auf seinem Computer im schreibgeschützten Modus betrachten.

2.1.2 Codebuch importieren/exportieren

Abgesehen vom Projekt-Import und -Export gibt es Szenarien, in denen lediglich das Codebuch importiert oder exportiert werden soll. QDAcity bietet hierfür bereits einen CSV-Export an, jedoch wird dieses Format von anderen QDA-Softwarelösungen nicht unterstützt. Daraus ergeben sich die folgenden funktionalen Anforderungen:

- **FA-06:** Der Nutzer kann ein Codebuch aus QDAcity exportieren.
- **FA-07:** Der Nutzer kann ein Codebuch in QDAcity importieren.
- **FA-08:** Der Nutzer kann ein Codebuch aus QDAcity in eine andere QDA-Software exportieren.
- **FA-09:** Der Nutzer kann ein Codebuch aus einer anderen QDA-Software in QDAcity importieren.

2.1.3 Exportformate für die Weiterverarbeitung

Während der Arbeit an den oben genannten Anforderungen äußerten aktive Nutzer von QDAcity zusätzliche Wünsche nach neuen Funktionen, die in den Umfang dieser Masterarbeit passen. Diese wurden daher in den Projektumfang aufgenommen. Daraus ergeben sich folgende zusätzliche funktionale Anforderungen:

- **FA-10:** Der Nutzer kann eine Übersicht aller Codierungen exportieren.
- **FA-11:** Der Nutzer kann eine Übersicht über die Anzahl aller Codierungen pro Dokument exportieren.

2.2 Nicht-funktionale Anforderungen

SOFTWARE PRODUCT QUALITY								
FUNCTIONAL SUITABILITY	PERFORMANCE EFFICIENCY	COMPATIBILITY	INTERACTION CAPABILITY	RELIABILITY	SECURITY	MAINTAINABILITY	FLEXIBILITY	SAFETY
FUNCTIONAL COMPLETENESS FUNCTIONAL CORRECTNESS FUNCTIONAL APPROPRIATENESS iso25000.com	TIME BEHAVIOUR RESOURCE UTILIZATION CAPACITY	CO-EXISTENCE INTEROPERABILITY	APPROPRIATENESS RECOGNIZABILITY LEARNABILITY OPERABILITY USER ERROR PROTECTION USER ENGAGEMENT INCLUSIVITY USER ASSISTANCE SELF-DESCRIPTIVENESS	FAULTLESSNESS AVAILABILITY FAULT TOLERANCE RECOVERABILITY	CONFIDENTIALITY INTEGRITY NON-REPUDIATION ACCOUNTABILITY AUTHENTICITY RESISTANCE	MODULARITY REUSABILITY ANALYSABILITY MODIFIABILITY TESTABILITY	ADAPTABILITY SCALABILITY INSTALLABILITY REPLACEABILITY	OPERATIONAL CONSTRAINT RISK IDENTIFICATION FAIL SAFE HAZARD WARNING SAFE INTEGRATION

Abbildung 2.1: Kriterien zur Qualität von Software (nach ISO 25010)

Die nicht-funktionalen Anforderungen orientieren sich an der ISO 25010 (siehe Abbildung 2.1), einem internationalen Standard für die Qualität von Softwareprodukten. Dieser Standard definiert verschiedene Qualitätsmerkmale, die Softwareprodukte erfüllen sollten, um eine hohe Benutzerzufriedenheit und eine gute Leistung zu gewährleisten.

- **NFA-01:** Die Daten sollen mit verschiedenen QDA-Softwarelösungen ausgetauscht werden können, wobei mindestens zwei interoperable Systeme unterstützt werden müssen. (Interoperabilität)
- **NFA-02:** Nur autorisierte Nutzer dürfen Projekte exportieren. (Sicherheit)
- **NFA-03:** Der Import- und Exportprozess soll auch bei größeren Projekten mit vielen Dokumenten und Codierungen schnell und effizient ablaufen. Die Antwortzeiten der Anwendung während dieses Vorgangs sollten akzeptabel sein und sich im Bereich von wenigen Sekunden bis Minuten bewegen. (Performance/Zeiteffizienz)
- **NFA-04:** Die Benutzeroberfläche für den Import und Export von Projekten und Codebüchern muss intuitiv und leicht verständlich gestaltet sein, sodass Nutzer mit minimalem Aufwand die gewünschten Daten importieren oder exportieren können. (Benutzbarkeit)
- **NFA-05:** Bei längeren Import- oder Exportvorgängen soll der Nutzer über den Fortschritt informiert werden, um Unklarheiten zu vermeiden. (Transparenz)
- **NFA-06:** Die Import- und Exportkomponenten sollen leicht wartbar und anpassbar sein, damit sie bei Bedarf schnell aktualisiert werden können. (Wartbarkeit)

- **NFA-07:** Neue Import- oder Exportfunktionen sollen einfach integrierbar sein. (Erweiterbarkeit)
- **NFA-08:** Die Import- und Exportfunktionalitäten müssen umfassend getestet werden, um sicherzustellen, dass ihre Funktionalität auch bei Änderungen an QDAcity erhalten bleibt. (Zuverlässigkeit)
- **NFA-09:** Im Falle von Fehlern während des Import- oder Exportvorgangs soll der Nutzer klare Informationen darüber erhalten, was schiefgelaufen ist. (Fehlerbehandlung)

3 Architektur

Im Folgenden wird die Architektur von QDAcity beschrieben, mit besonderem Fokus auf der Speicherung von und dem Zugriff auf Daten. Dabei wird erläutert, in welcher Form die Daten gespeichert sind.

3.1 QDAcity

QDAcity ist eine cloudbasierte QDA-Softwareanwendung, die aus zahlreichen Komponenten besteht. Für diese Arbeit sind insbesondere die folgenden von Relevanz.

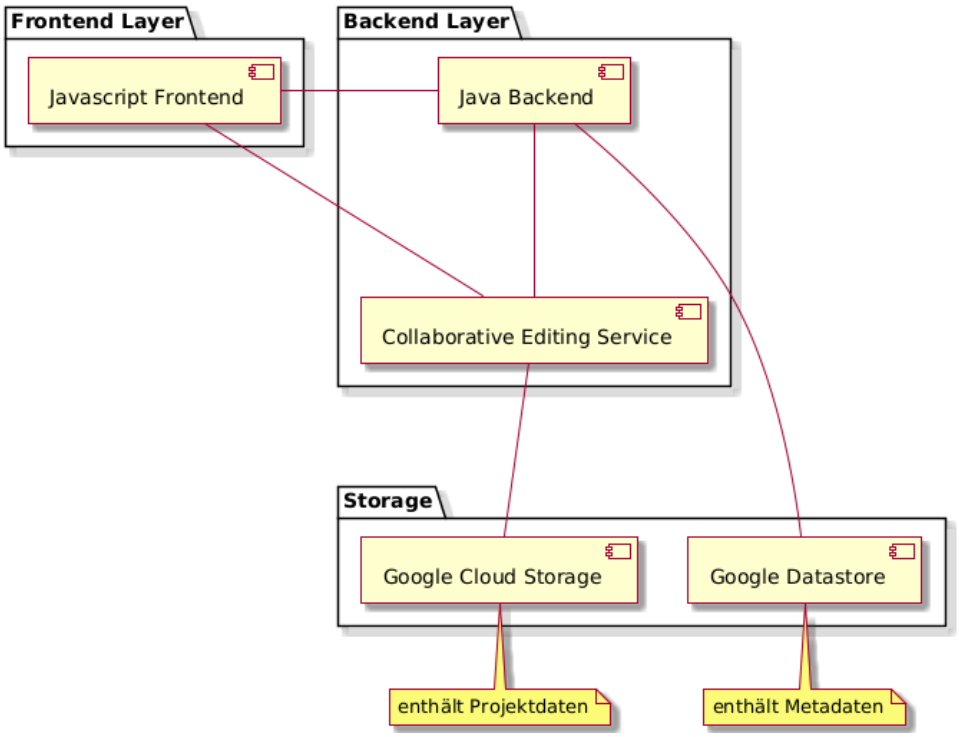

Abbildung 3.1: QDAcity grobe Architektur

QDAcity ist im Wesentlichen folgendermaßen strukturiert:

- **Frontend:** Das Frontend basiert auf JavaScript und verwendet das React-Framework, um dynamische und interaktive Benutzeroberflächen zu erstellen.
- **Backend:** Im Hintergrund arbeitet ein Java-Backend, das die Verwaltung der Daten übernimmt. Dieses Backend ist die einzige Komponente, die direkten Zugriff auf die Datenbank hat. Es speichert unter anderem die Metadaten von Datenobjekten im Google Datastore.
- **Collaborative Editing Service (CES):** Zusätzlich gibt es den Collaborative Editing Service (CES), der es Nutzern ermöglicht, gleichzeitig an einem Projekt zu arbeiten. Kollaboratives Arbeiten bedeutet hierbei die gleichzeitige Bearbeitung oder allgemeine Zusammenarbeit an einem gemeinsamen Projekt.

3.1.1 Unterstützte Datenstrukturen in QDAcity

Da diese Arbeit den Austausch von Daten behandelt, wird im Folgenden die für den Datenaustausch relevante Datenstruktur von QDAcity erläutert.

QDAcity unterstützt die folgenden Datenstrukturen:

- **Projekt:** Ein übergeordnetes Konstrukt, das alle relevanten Elemente umfasst und an dem Forscher arbeiten können.
- **Codesystem (Codebuch):** Das Projekt enthält ein Codesystem, auch bekannt als Codebuch, das sämtliche Codes organisiert und speichert.
- **Dokumente:** QDAcity unterstützt sowohl Text- als auch PDF-Dokumente. Zusätzlich können Audiotranskripte erstellt und als Textdokumente gespeichert werden.
- **Codierungen:** Codierungen entstehen durch die Selektion von Text oder Bereichen in Dokumenten und das Anwenden von Codes. QDAcity unterscheidet zwischen drei Arten von Codierungen:
  - **TextCoding:** Selektion von Text in einem Textdokument.
  - **PDFAreaCoding:** Auswahl von Flächen in einem PDF-Dokument.
  - **PDFTextCoding:** Markierung von Text innerhalb eines PDF-Dokuments.

Diese Datenstrukturen bilden die Grundlage für den Datenaustausch in QDAcity und sind essenziell für die Weiterverarbeitung sowie die Analyse von Forschungsdaten.

3.1.2 Speicherung der Datenstrukturen

Die Speicherung der Daten erfolgt wie folgt:

- **Projekt:** Beim Erstellen eines Projekts wird ein entsprechender Eintrag in der Datenbank erstellt und eine Ydoc-Datei (binäre Yjs-Daten) im Google Cloud Storage (GCS) angelegt. Diese Datei enthält die Projektdaten. Codes und Codierungen werden über den CES im Projekt-Ydoc gespeichert.
- **Dokumente:** Beim Hochladen eines Dokuments wird ebenfalls ein Datenbankeintrag erstellt. Der Inhalt der Dokumente wird in GCS gespeichert:
  - Für Textdokumente wird ein Ydoc erstellt, das für das kollaborative Bearbeiten mit dem Yjs-Editor benötigt wird.
  - Der Inhalt von PDF-Dokumenten wird direkt in GCS hochgeladen.

4 Design und Implementierung

4.1 REFI-QDA

REFI-QDA ist ein Austauschformat, das von der Rotterdam Exchange Format Initiative (REFI) entwickelt wurde. Ziel des Formats ist es, den Austausch von Projekten und Codebüchern zwischen verschiedenen QDA-Softwarelösungen zu ermöglichen und zu vereinfachen. Das REFI-QDA-Standard wurde in Zusammenarbeit von Entwicklern mehrerer QDA-Softwarelösungen konzipiert und basiert auf zwei zentralen Datenmodellen: einem für das Codebuch (siehe Abbildung 4.1) und einem für Projekte (siehe Abbildung 4.2). Diese Modelle werden durch XML-Schemata definiert, die die Struktur der für den Austausch verwendeten XML-Dateien festlegen.

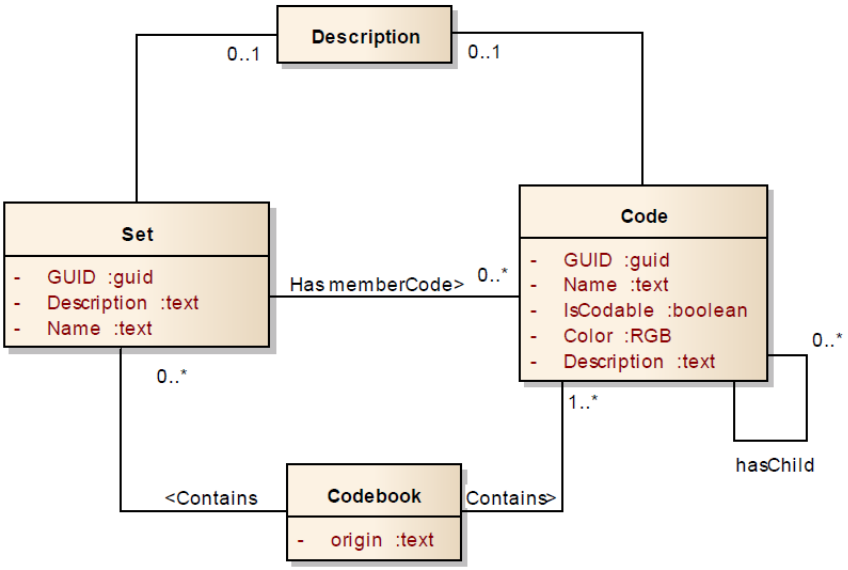

Abbildung 4.1: Codebuch-Datenmodell (van Blommestein, 2019)

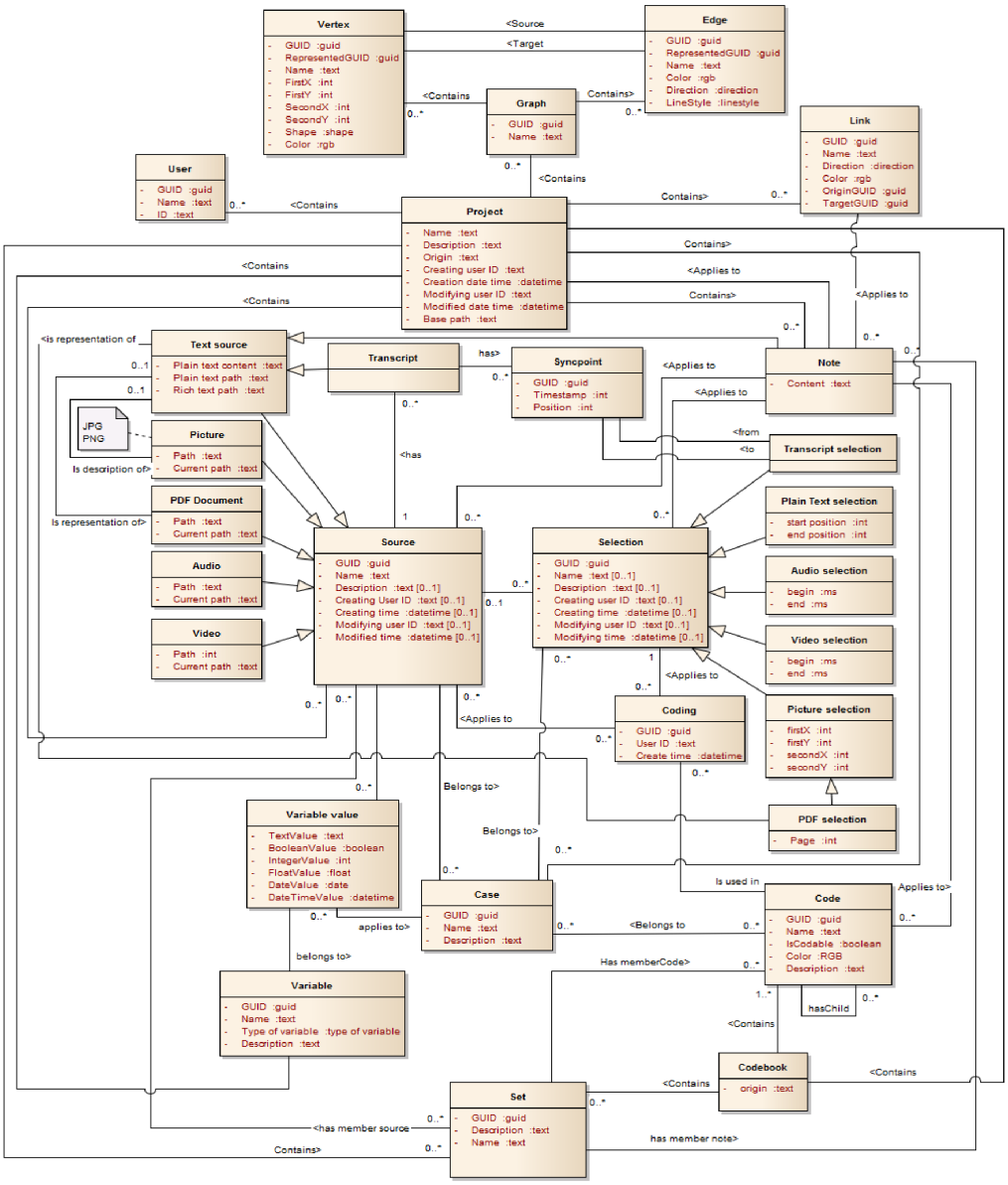

Abbildung 4.2: Projekt-Datenmodell (van Blommestein, 2019)

Struktur des REFI-QDA-Formats:

- **Codebuch-Austausch:** Ein Codebuch wird in einer XML-Datei mit der Endung `.qdc` gespeichert.
- **Projekt-Austausch:** Projekte werden als Zip-Archive mit der Endung `.qdpx` gespeichert. Diese Archive enthalten:
  - Eine XML-Datei mit der Endung `.qde`, die die Projektdaten enthält.
  - Einen `source`-Ordner, der die zugehörigen Dokumente beinhaltet. Dieser Ordner darf keine Unterordner enthalten, und die Dateien müssen anhand ihrer GUID (wie in der `.qde`-Datei angegeben) eindeutig benannt sein.

Einschränkungen und Vorteile
Es ist jedoch zu beachten, dass nicht jede QDA-Software alle Elemente des Datenmodells unterstützt, da die Funktionen der verschiedenen Softwarelösungen variieren können. Die Elemente, die im REFI-QDA-Standard enthalten sind, werden von mindestens zwei QDA-Softwarelösungen unterstützt, was ihre Aufnahme in das Austauschformat begründet. Dennoch kann es vorkommen, dass nicht alle Elemente von jeder QDA-Software vollständig unterstützt werden, was zu einem Verlust von Daten beim Austausch von Codebüchern oder Projekten führen kann.

Trotz dieser Einschränkungen wurde das Format bewusst entwickelt, um den Austausch zwischen QDA-Softwarelösungen zu erleichtern, und ist inzwischen in vielen QDA-Softwarelösungen implementiert. In diesem Zusammenhang soll das REFI-QDA-Format auch in QDAcity integriert werden. Obwohl QDAcity nicht alle Elemente des Datenmodells unterstützt, wurde das vollständige Datenmodell übernommen, da in Zukunft die Möglichkeit besteht, dass QDAcity diese Funktionen hinzufügen könnte.

Implementierung in QDAcity
Das REFI-QDA-Datenmodell wurde in das Java-Backend von QDAcity integriert. Für jede Klasse im Datenmodell wurde eine Methode zur Generierung eines XML-Strings entwickelt, die das XML rekursiv aufbaut. Da XML eine hierarchische Struktur besitzt, wird der Prozess mit dem Wurzelknoten gestartet, der die gleiche Methode für seine Kindknoten aufruft. Diese erzeugen wiederum ihre XML-Tags mit den entsprechenden Attributen und Inhalten und geben sie an die übergeordnete Instanz zurück.

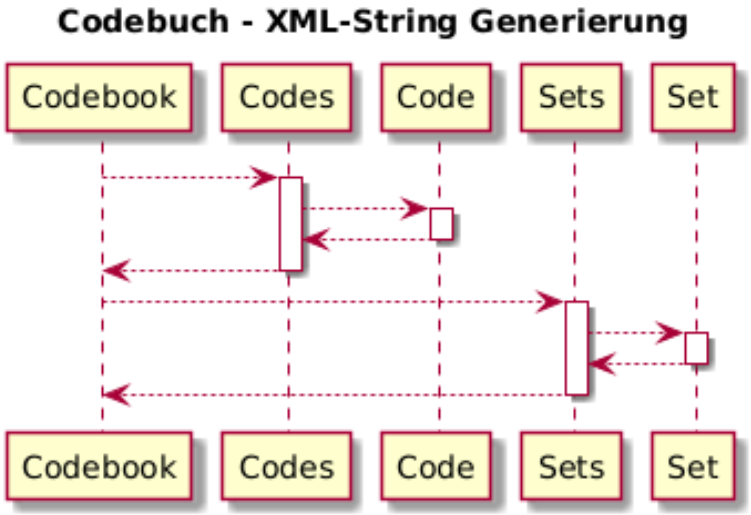

Abbildung 4.3: Prozess zur Erstellung des XML-Strings für den Codebuch-Export

Ein Beispiel für diesen Prozess ist in Abbildung 4.3 dargestellt, die die Generierung eines XML-Strings für den Codebuch-Export veranschaulicht.

Beim Codebuch-Export beginnt die Generierung des XML-Strings mit dem Wurzelknoten `Codebook`. Dieser erzeugt zunächst seinen eigenen XML-Tag. Anschließend ruft er die Methode für seine direkten Kindknoten `Codes` und `Sets` auf, welche wiederum ihre eigenen XML-Tags generieren. Für die jeweiligen Kindknoten `Code` und `Set` wird dieselbe Methode rekursiv aufgerufen, um die entsprechenden Tags mit Inhalten zu erzeugen und zurückzugeben.

In der tatsächlichen Implementierung gibt es jedoch eine Abweichung: `Codes` und `Sets` existieren im Datenmodell nicht als eigenständige Klassen. Obwohl sie im XML-Schema als Tags definiert sind, werden sie im Modell als Arrays innerhalb von `Codebook` repräsentiert. Deshalb erfolgt die Generierung der XML-Tags für `Codes` und `Sets` direkt innerhalb von `Codebook`. Trotz dieser Anpassung bleibt der rekursive Ablauf der XML-Generierung unverändert.

4.2 Export

Vor dem Refactoring dieser Arbeit wurden die Daten direkt im Frontend zusammengeführt und exportiert. Mit der neuen Architektur werden die Daten nun vollständig im Backend verarbeitet und anschließend als String an das Frontend übergeben, das diese als Datei bereitstellt. Der überarbeitete Exportprozess (Abbildung 4.4) ist wie folgt strukturiert:

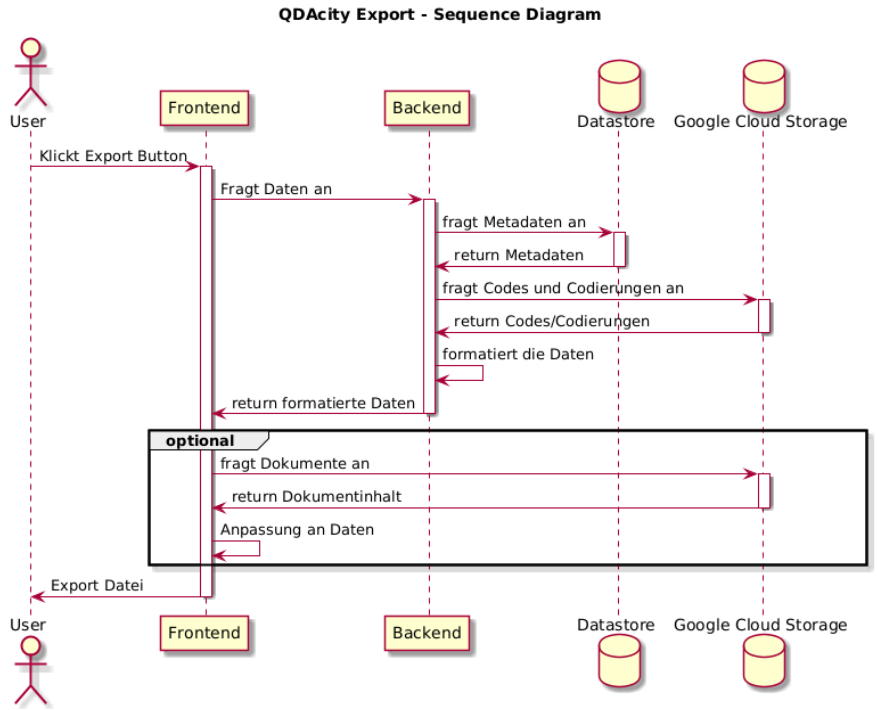

Abbildung 4.4: Export-Sequenzdiagramm

Der Exportprozess beginnt, indem der Nutzer auf den Export-Button klickt. Daraufhin sendet das Frontend eine Anfrage an das Java-Backend. Nach dem Refactoring wurde im Java-Backend die Klasse `ExportEndpoint` eingeführt, die für die Verarbeitung der Exportanfragen zuständig ist. Je nach gewünschtem Exporttyp wird die entsprechende Methode aus der Klasse `ExportEndpoint` aufgerufen (siehe Tabelle 4.1). Diese Methoden empfangen folgende Parameter:

- `projectID`: die ID des aktuellen Projekts,
- `projectType`: den Typ des Projekts,
- den aktuellen Nutzer (`AuthenticatedUser`)
- `isCompact`: Wird nur für den CSV-Codings-Summary-Export verwendet, um zwischen den beiden Versionen zu wechseln.

Mit diesen Parametern wird geprüft, ob die Anfrage autorisiert ist. Nur wenn ein autorisierter Nutzer den Export gestartet hat, verarbeitet das Backend die Anfrage weiter. Dazu ruft das Java-Backend die für den Export benötigten Daten entweder vom CES oder aus der Datenbank ab.

Sobald alle erforderlichen Daten vorliegen, konvertiert das Backend sie in das gewünschte Exportformat. Die fertig formatierten Daten werden anschließend als String an das Frontend zurückgegeben. In bestimmten Fällen können zusätzliche Anpassungen an den Daten im Frontend vorgenommen werden, um das Exportformat zu vervollständigen.

Schließlich wird eine signierte URL[1] generiert, mit der die Daten als Datei zum Herunterladen bereitgestellt werden. Diese URL garantiert einen sicheren und zeitlich begrenzten Zugriff auf die Exportdaten.

Methode	**Parameter**
`csvCodebook`	`projectID: Long, projectType: ProjectType, user: AuthenticatedUser`
`qdcCodebook`	`projectID: Long, projectType: ProjectType, user: AuthenticatedUser`
`qdeProject`	`projectID: Long, projectType: ProjectType, user: AuthenticatedUser`
`csvCodematrix`	`projectID: Long, projectType: ProjectType, user: AuthenticatedUser`
`csvCodingSummary`	`projectID: Long, projectType: ProjectType, user: AuthenticatedUser, isCompact: Boolean`

Tabelle 4.1: Methoden der `ExportEndpoint`-Klasse mit Parametern

4.2.1 Adapter

Um die bestehenden Daten in das richtige Format für den Export zu überführen, wurden zwei Java-Klassen entwickelt, die jeweils für das Abrufen und Formatieren der Daten verantwortlich sind.

Für den CSV-Export wurde die Java-Klasse `CSVConverter` erstellt, die die benötigten Daten verarbeitet und im entsprechenden CSV-Format aufbereitet.

Für den Export der REFI-QDA-Formate wurde die Klasse `REFIQDAConverter` implementiert, die die Daten in das entsprechende Format konvertiert.

Diese Klassen enthalten für jeden Exporttyp eine Methode, die die entsprechenden Daten abruft und anschließend formatiert. Diese Methoden werden dann vom `ExportEndpoint` aufgerufen.

[1] https://cloud.google.com/storage/docs/access-control/signed-urls

4.2.2 Codebuch-Export

QDAcity verfügte bereits über ein eigenes Format für den Codebuch-Export, bei dem die Daten als CSV-Datei ausgegeben werden. Um jedoch den Datenaustausch mit anderen QDA-Softwareanwendungen zu ermöglichen, wurde zusätzlich das QDC-Format von REFI-QDA implementiert.

4.2.2.1 CSV-Codebuch-Export

Der CSV-Codebuch-Export generiert eine CSV-Datei, die für jeden Code die folgenden Informationen enthält:

- **Code-Name:** Der Name des Codes.
- **Code-Pfad:** Die hierarchische Struktur, in der der Code organisiert ist.
- **Definition:** Eine ausführliche Beschreibung des Codes.
- **ShortDefinition:** Eine kurze, prägnante Beschreibung des Codes.
- **WhenToUse:** Beschreibung, wann der Code verwendet werden soll.
- **WhenNotToUse:** Beschreibung, wann der Code nicht verwendet werden soll.
- **Example:** Ein Beispiel, das den Anwendungsbereich des Codes illustriert.
- **Time used:** Die Anzahl der Codierungen, die den Code verwenden.
- **Code-ID:** Eine eindeutige Identifikationsnummer des Codes.

Das Abrufen und Formatieren der Daten erfolgt in der Klasse `CSVConverter`. Die für den Export benötigten Daten werden aus dem CES bezogen. Die Klasse `CesCodeController` liefert eine Liste der `Code`-Objekte des aktuellen Projekts, während die Klasse `CesCodingController` die Codierungen des Projekts zurückgibt.

Die Code-ID und der Code-Name sind als Attribute im `Code`-Objekt enthalten. Die Definition, ShortDefinition, WhenToUse, WhenNotToUse und Example werden hingegen in der `CodeBookEntry`-Klasse gespeichert, welche als Attribut im `Code`-Objekt eingebunden ist.

Der Code-Pfad wird auf Basis der `parentId` des `Code`-Objekts generiert. Die `parentId` stellt die Code-ID des übergeordneten Codes dar. Iterativ wird über die `parentId` der Codes der jeweilige übergeordnete Code ermittelt, bis der Wurzel-Code erreicht ist, der keinen übergeordneten Code besitzt.

Die Anzahl der Codierungen wird durch Zählen ermittelt. Hierfür wird das Attribut `localCodeId` der Codierungen verwendet, das mit der Code-ID des zugehörigen `Code`-Objekts übereinstimmt. Für jeden Code wird gezählt, wie oft dessen Code-ID in den Codierungen vorkommt.

Alle diese Informationen werden mithilfe der Hilfsklasse `CodeBookCSV` verarbeitet. Für jeden Code wird eine Instanz von `CodeBookCSV` erstellt, die die relevanten Daten speichert und daraus eine entsprechende CSV-Zeile generiert.

Abschließend werden alle CSV-Zeilen der `CodeBookCSV`-Instanzen alphabetisch nach dem Code-Pfad sortiert. Die sortierten CSV-Zeilen werden anschließend zusammen mit dem Header zu einem String zusammengeführt, der an das Frontend zurückgegeben wird.

4.2.2.2 QDC-Codebuch-Export

Der QDC-Codebuch-Export richtet sich nach dem REFI-QDA-Standard und generiert eine XML-Datei mit der Endung .qdc. Das Codebuch umfasst Codes und Sets, wobei Sets benannte Sammlungen von Quellen, Codes und/oder Notizen darstellen. Da Sets in QDAcity nicht unterstützt werden, werden sie im Export weggelassen.

Für jeden Code werden die folgenden Informationen exportiert:

- **GUID:** Eine global eindeutige Identifikationsnummer des Codes.
- **Name:** Der Name des Codes.
- **Code-Farbe:** Die Farbe des Codes.
- **isCodable:** Gibt an, ob der Code für die Codierung verwendet werden kann.
- **Description:** Eine optionale Beschreibung des Codes.
- **Children:** Alle Codes, die diesem Code untergeordnet sind

Für REFI-QDA-Formate wird die Klasse `REFIQDAConverter` verwendet. Diese Klasse übernimmt das Abrufen und Formatieren der benötigten Daten. Wie beim CSV-Codebuch-Export werden die benötigten Daten aus dem CES bezogen. Für den QDC-Codebuch-Export wird ausschließlich die von der Klasse `CESCodeController` zurückgelieferte Liste von `Code`-Objekten benötigt.

Ähnlich wie beim CSV-Codebuch-Export wird auch beim QDC-Codebuch-Export eine Hilfsdatenstruktur verwendet. Der entscheidende Unterschied besteht jedoch darin, dass beim QDC-Export ein vollständiges Datenmodell (Abbildung 4.1) zum Einsatz kommt, anstelle einer einzelnen Hilfsklasse.

Dem XML-Schema (siehe Anhang B) ist zu entnehmen, dass `Codebook` den Wurzelknoten der XML-Datei darstellt. Wie in Kapitel 4.1 beschrieben, beginnt die Generierung des XML-String am Wurzelknoten.

Zunächst wird in der Klasse `REFIQDAConverter` ein `Codebook`-Objekt erstellt. Dieses wird mit REFI-QDA-Code-Objekten befüllt, die aus den `Code`-Objekten generiert werden.

Die GUID wird bei der Erstellung des REFI-QDA-Code-Objekts generiert, da der REFI-QDA-Standard ein bestimmtes Format dafür vorschreibt. Der Name und die Farbe des Codes sind als Attribute im `Code`-Objekt enthalten. Da alle Codes in QDAcity für die Codierung verwendet werden können, wird das Attribut `isCodable` standardmäßig auf `true` gesetzt. Da es keine eindeutige Beschreibung für das Attribut `Description` gibt, wird dieses zunächst weggelassen.

Im Gegensatz zu QDAcity wird beim REFI-QDA-Standard der untergeordnete Code anstelle des übergeordneten Codes hinterlegt. Aus diesem Grund werden bei der Erstellung der REFI-QDA-Code-Objekte zunächst die REFI-QDA-Code-Objekte zusammen mit der Code-ID des `Code`-Objekts in einer Map gespeichert. Sobald für alle `Code`-Objekte ein entsprechendes REFI-QDA-Code-Objekt erstellt wurde, wird erneut über die `Code`-Objekte iteriert. Dabei wird anhand der `parentId` das entsprechende REFI-QDA-Code-Objekt ermittelt, und das REFI-QDA-Code-Objekt, das mit der Code-ID verknüpft ist, wird als Kindknoten hinzugefügt.

Abschließend ruft die Klasse `REFIQDAConverter` die Methode zur Generierung des XML-Strings für den Codebuchaustausch in der `Codebook`-Klasse auf. Nachdem der XML-String von `Codebook` zurückgegeben wurde, wird er gegen das XML-Schema validiert. Ist der String valide, wird er an das Frontend zurückgegeben.

4.2.3 Codematrix

Die Codematrix stellt eine Codierungsfrequenz dar, in der die X-Achse die Dokumente und die Y-Achse die Codes repräsentiert. Jede Zelle ist die Anzahl der Codierungen, das heißt, wie oft ein bestimmter Code in einem bestimmten Dokument verwendet wurde. Die Codematrix wird zu diesem Zweck im CSV-Format dargestellt.

Für diesen Export werden folgende Informationen benötigt:

- Namen der Dokumente
- Namen der Codes
- Anzahl der Codierung eines Codes pro Dokument

Das Abrufen und Formatieren der Daten wird in der Klasse `CSVConverter` durchgeführt. Die für den Export benötigten Daten werden sowohl aus dem CES als auch aus der Datenbank abgerufen. Die Klasse `CesCodeController` stellt eine Liste der `Code`-Objekte des aktuellen Projekts zur Verfügung, während die Klasse `CesCodingController` die Codierungen des Projekts liefert. Darüber hinaus stellt die Klasse `DocumentController` eine Liste der Dokumente bereit, die aus der Datenbank stammen.

Zunächst wird der Header erstellt, der sich durch die Verkettung aller Dokumentnamen zusammensetzt. Dem Header wird das Wort „Code“ vorangestellt, da die Codes auf der Y-Achse dargestellt werden.

Beim Erstellen des Headers wird jedem Dokument ein Index von 0 bis zur Anzahl der Dokumente minus 1 zugewiesen, der dessen Position widerspiegelt.

Um die verbleibenden CSV-Zeilen zu generieren, muss für jeden Code ermittelt werden, wie oft seine Codierung in jedem Dokument angewandt wurde. Hierfür wird ein Array erstellt, dessen Länge der Anzahl der Dokumente entspricht. Mit diesem Array wird die Anzahl der Codierungen des jeweiligen Codes pro Dokument erfasst.

Für jedes `Code`-Objekt wird über alle Codierungen iteriert. Dabei wird die Code-ID des `Code`-Objekts mit der `localCodeId` der jeweiligen Codierung verglichen. Wenn diese übereinstimmen, wird der Index des entsprechenden Dokuments anhand der `documentId` der Codierung ermittelt und der Wert im Array an der entsprechenden Stelle um 1 erhöht. Auf diese Weise wird die Anzahl der Codierungen eines Codes pro Dokument gezählt. Die Werte im Array werden anschließend verkettet, der Name des aktuellen Codes wird vorangestellt und die resultierende Zeile wird dem CSV-String hinzugefügt.

Hier ein Beispiel: Angenommen, wir haben 5 Dokumente für den Code `Code System` und 3 Codierungen dieses Codes. Die erste Codierung befindet sich in Dokument 1, die zweite in Dokument 4 und die dritte in Dokument 0. In diesem Fall wird das Integer-Array der Länge der Dokumente wie folgt gefüllt:

Schritt 1:	0	0	0	0	0
Schritt 2:	0	1	0	0	0
Schritt 3:	0	1	0	0	1
Schritt 4:	1	1	0	0	1

Abbildung 4.5: Beispiel: Array zur Erfassung der Anzahl der Codierungen pro Dokument

Die CSV-Zeile würde in diesem Fall folgendermaßen aussehen:

```
"Code System",1,1,0,0,1\n
```

Nachdem die Code-Matrix als CSV-String erstellt wurde, wird sie an das Frontend zurückgegeben.

4.2.4 Coding Summary

Die Coding-Summary stellt eine strukturierte Übersicht aller Codierungen dar. Dabei werden die Vorschautexte der Codierungen jeweils mit den zugehörigen Code-Pfaden verknüpft und in einer CSV-Datei ausgegeben.

Es existieren zwei Varianten der Darstellung:

- **Normale Version**: Jede Codierung wird mit dem vollständigen Code-Pfad und der dazugehörigen Vorschau angezeigt.
- **Kompakte Version**: Bei dieser Variante werden die Vorschautexte der Codierungen zusammengefasst und den jeweiligen Codes zugeordnet. Dabei wird die Hierarchie des Codebuchs abgebildet.

Da die Daten in einer CSV-Datei ausgegeben werden, erfolgt das Abrufen und Formatieren der Informationen in der Klasse `CSVConverter`. Dabei wird die Hilfsklasse `CodeBookCSV` verwendet, da insbesondere für die kompakte Variante die Codierungen zu den jeweiligen Codes zusammengefasst werden müssen. Die benötigten Daten werden vom CES bereitgestellt. Die Klasse `CesCodeController` liefert eine Liste der `Code`-Objekte des aktuellen Projekts, während die Klasse `CesCodingController` die zugehörigen Codierungen zurückgibt.

Für jeden Code wird eine Instanz der Klasse `CodeBookCSV` erstellt. Diese Instanz wird mit dem generierten Code-Pfad sowie den Codierungen, die diesem Code zugeordnet sind, befüllt.

Anschließend werden die `CodeBookCSV`-Instanzen nach ihrem Code-Pfad alphabetisch sortiert, um die Hierarchie des Codebuchs korrekt abzubilden.

In der normalen Version wird kein Header erzeugt. Für alle Codes wird der Code-Pfad abgerufen und mit dem CSV-Separator konkateniert. Danach wird für jede Codierung eines Codes der konkatenierte Code-Pfad mit der entsprechenden Vorschau verknüpft, um eine vollständige CSV-Zeile zu bilden. Die Gesamtheit aller CSV-Zeilen wird dann zu einem CSV-String zusammengefügt, der an das Frontend zurückgegeben wird.

In der kompakten Version werden die CSV-Zeilen zuerst erstellt, der Header wird erst am Ende hinzugefügt. Für alle Codes wird, abhängig von der Länge des Code-Pfads, der Anfang der CSV-Zeile leer gelassen, das heißt, die Zellen am Anfang sind leer. Für jedes Element im Code-Pfad werden zwei CSV-Separatoren eingefügt, was zu zwei leeren Zellen pro Code im Pfad führt. Nur für den Wurzelknoten „Code System“ wird eine Zelle freigelassen, unter der Annahme, dass der Code „Code System“ nicht verwendet wird, da er die Wurzel des Codebuchs darstellt und somit nicht als vom Nutzer gewünschter Code betrachtet wird. An diese leere CSV-Zeile wird dann der aktuelle Code angehängt, gefolgt von einer Sammlung aller Vorschautexte der zugehörigen Codierungen. Es ist anzumerken, dass nicht alle Codierungstypen eine Vorschau besitzen. Daher wurde in der Klasse `CodeBookCSV` eine Methode erstellt, die die Codierungen ohne Vorschau zählt und anstelle der Vorschau folgenden Text ausgibt:

```
X {Codierungstyp} without preview.
```

Wobei X die Anzahl der Codierungen des Codierungstyps ohne Vorschau darstellt.

Nachdem die CSV-Zeilen für alle Codes erstellt wurden, wird auch die Länge des längsten Code-Pfades ermittelt und gespeichert. Anhand dieser Länge wird der Header erstellt. Für den Wurzelknoten „Code System“ wird eine leere Zelle hinzugefügt, gefolgt von den Einträgen „Code Level X“ und „Quotes Collection X“ im Header. Dabei wird der Wert von X von 1 bis zur Länge des längsten Code-Pfades minus 1 wiederholt.

Anschließend wird der Header mit allen CSV-Zeilen kombiniert und als CSV-String an das Frontend zurückgegeben.

4.2.5 Projekt-Export

Der QDPX-Projekt-Export folgt dem REFI-QDA-Standard und erstellt eine Archivdatei im ZIP-Format mit der Endung .qdpx. Dieses Archiv enthält eine XML-Datei mit der Endung .qde, die dem QDC-Codebuch-Export ähnelt, jedoch mehr Informationen als nur das Codebuch speichert. Zusätzlich umfasst das Archiv einen `source`-Ordner, in dem die Dokumente des Projekts abgelegt sind. Im Gegensatz zu anderen Exporten werden hier mehrere Dateien benötigt. Dabei übernimmt das Backend die Erstellung der QDE-Datei, während das Frontend das QDPX-Archiv erzeugt und die Dokumente für den `source`-Ordner bereitstellt.

Die QDE-Datei, die das Projekt repräsentiert, kann eine Vielzahl von Informationen enthalten. Das Datenmodell für das Projekt (Abbildung 4.2) ist daher deutlich umfangreicher als das Datenmodell des Codebuch-Exports im REFI-QDA-Standard.

In QDAcity werden jedoch nicht alle Elemente des Projekt-Exports unterstützt. Daher werden nur das Codebuch, die Codierungen, die Dokumente und die Nutzer exportiert. Die Struktur des Codebuchs ist dieselbe wie beim QDC-Codebuch-Export und enthält die Codes. Im Folgenden werden daher nur die Codierungen, Dokumente und Nutzer behandelt.

4.2.5.1 Erstellung des XML-Strings

Die Erstellung des XML-Strings für die QDE-Datei erfolgt durch die Klasse `REFIQDAConverter`. Diese ist für das Abrufen der erforderlichen Daten sowie deren Formatierung gemäß dem REFI-QDA-Standard verantwortlich. Der Prozess umfasst die Generierung eines hierarchisch strukturierten XML-Strings, wobei die Klasse `Project` als Wurzelknoten fungiert.

Datenquellen und Verarbeitung
Die benötigten Daten werden aus unterschiedlichen Quellen bezogen:

- **Codes und Codierungen:** Diese Informationen werden über den CES bereitgestellt.
- **Dokumente und Nutzer:** Diese Daten werden aus der Datenbank extrahiert. Dabei stellt der `MemberController` die am Projekt beteiligten Nutzer bereit, während der `DocumentController` eine Liste der im Projekt enthaltenen Dokumente liefert.

Der Prozess beginnt mit der Erstellung des Wurzelknotens `Project`, der lediglich den Projektnamen enthält. Dieser wird mithilfe des `ProjectController` aus der Datenbank abgerufen.

Integration der Nutzerdaten
Für jeden Projektbeteiligten wird ein `User`-Objekt erstellt, das den Namen und die eindeutige ID des Nutzers als Attribute enthält. Diese Objekte werden anschließend dem `Project`-Objekt hinzugefügt.

Verarbeitung der Dokumentdaten
Im REFI-QDA-Standard werden Dokumente mithilfe der Klasse `Source` repräsentiert, die verschiedene Dokumenttypen unterstützt. Da QDAcity ausschließlich Text- und PDF-Dokumente verarbeitet, kommen in diesem Kontext die Klassen `TextSource` und `PDFSource` zum Einsatz.

Die Erstellung der `Source`-Objekte basiert auf den folgenden Attributen:

- **Name des Dokuments:** Dieser wird direkt aus den entsprechenden Datenbankobjekten extrahiert und repräsentiert den Titel des Dokuments.
- **Dateipfad:** Der standardmäßige Dateipfad verweist auf den Ordner `source` innerhalb des später zu exportierenden QDPX-Archivs. Externe Speicherorte werden nicht unterstützt.
- **Dokument-ID:** Die Dokument-ID wird ebenfalls aus den Datenbankobjekten extrahiert. Sie dient als Platzhalterwert, der im Frontend verwendet wird, um die `Source`-Objekte den tatsächlichen Dokumenten korrekt zuzuordnen.

Zusätzlich erfordert der REFI-QDA-Standard, dass für PDF-Dokumente eine Klartext-Variante bereitgestellt wird, welche durch die Klasse `Representation` dargestellt und ebenfalls im Ordner `source` abgelegt wird. Während der Erstellung der `Source`- und `Representation`-Objekte werden Platzhalterwerte für die Dateipfade verwendet, die später im Frontend durch die tatsächlichen Werte ersetzt werden. Zur Sicherstellung der korrekten Zuordnung von Codierungen zu den Dokumenten werden die `Source`-Objekte in einer Map gespeichert, wobei die Dokument-ID als Schlüssel dient.

Verarbeitung der Codes und Codierungen
Das Codebuch und die Codes werden entsprechend dem QDC-Codebuch-Format verarbeitet. Hierbei werden die erstellten `Code`-Objekte in einer Map gespeichert, wobei die Code-ID der QDAcity-Code-Objekte als Schlüssel dient, um sie später abrufen zu können.

Im Rahmen des REFI-QDA-Standards werden Codierungen durch eine Kombination der Klassen `Selection` und `Coding` dargestellt. Dabei ist die Klasse `Coding` der `Selection` untergeordnet und spezifiziert, welcher Code auf die jeweilige Selektion angewendet wurde. Die `Selection`-Klasse beschreibt die konkrete Auswahl im Dokument.

Da QDAcity nur drei Codierungsarten unterstützt, werden in diesem Kontext lediglich die Klassen `PlainTextSelection` und `PDFSelection` benötigt:

- `PlainTextSelection`: Für Textselektionen in Text- und PDF-Dokumenten.
- `PDFSelection`: Für Flächenselektionen in PDF-Dokumenten.

Für jede Codierung wird ein `Selection`- und ein `Coding`-Objekt erstellt, wobei `Coding` auf das entsprechende `Code`-Objekt verweist. Mit der `localCodeId` der Codierung kann das passende `Code`-Objekt aus der Map abgerufen und dem neu erstellten `Coding` zugeordnet werden. Dieses `Coding`-Objekt wird schließlich der entsprechenden `Selection` übergeben.

Die zur Erstellung der `Selection`-Objekte benötigten Informationen umfassen:

- `PlainTextSelection`: Start- und Endposition der Selektion im Textdokument.
- `PDFSelection`: Seitennummer sowie die X- und Y-Koordinaten des rechteckigen Bereichs, der die Selektion im PDF-Dokument beschreibt.

Da diese Informationen im Backend nicht direkt verfügbar sind, werden Platzhalterwerte verwendet, die später im Frontend ersetzt werden.

`Selection`-Objekte müssen außerdem den jeweiligen `Source`-Objekten zugeordnet werden. Hierfür wird die Dokument-ID der Codierung verwendet, um das passende `Source`-Objekt aus der Map abzurufen.

Validierung und Übergabe
Nach Abschluss der Erstellung aller relevanten Objekte erfolgt die Generierung des XML-Strings durch das `Project`-Objekt. Dabei wird der erstellte XML-String validiert, indem er gegen das zugrunde liegende XML-Schema geprüft wird. Abschließend wird der validierte XML-String an das Frontend übergeben.

4.2.5.2 Archiverstellung sowie Weiterverarbeitung der QDE-Datei

Nachdem das Frontend den XML-String empfangen hat, wird ein Zip-Archiv erstellt, um die zu exportierenden Dokumente im weiteren Prozess zu integrieren. Innerhalb dieses Archivs wird ein Ordner mit der Bezeichnung `source` angelegt, in dem die Dokumente gespeichert werden sollen. Anschließend erfolgt eine Iteration über alle relevanten Dokumente. Dabei wird jedes Dokument aus dem GCS abgerufen und die Platzhalterwerte im XML-String werden mithilfe des jeweiligen Dokumenteninhalts ersetzt. Dieser Schritt unterscheidet sich je nach Dokumententyp, da Text- und PDF-Dokumente unterschiedlich behandelt werden müssen.

Ersetzung der Platzhalterwerte für Dateipfade
Um den Platzhalterwert des Dateipfads in den `Source`-Objekten zu ersetzen, wird für jedes Dokument anhand der Dokument-ID die entsprechende `Source` ermittelt und deren GUID ausgelesen. Die GUID dient anschließend als Name für das exportierte Dokument im `source`-Ordner, wodurch eine eindeutige Zuordnung sichergestellt wird.

Verarbeitung von PDF-Dokumenten
Für ein PDF-Dokument wird der Inhalt zunächst aus dem GCS abgerufen, da das Dokument dort gespeichert ist. Anschließend wird es im `source`-Ordner des QDPX-Archivs abgelegt. Danach erfolgt eine Bearbeitung der Codierungen, da diese zu diesem Zeitpunkt noch Platzhalterwerte für die `Selection`-Objekte enthalten. PDF-Dokumente können dabei zwei verschiedene Arten von Codierungen aufweisen: `PDFSelection` und `PlainTextSelection`.

Die `PDFSelection` erfordert Informationen über die Seite, auf der sich die Codierung befindet, sowie die X- und Y-Koordinaten im PDF-Dokument. QDAcity speichert diese Informationen in den Codierungen, jedoch gibt es Unterschiede in der Art und Weise, wie diese Werte im REFI-QDA-Standard und in QDAcity dargestellt werden. Während QDAcity bei der Seitenzahl bei 1 beginnt, beginnt der REFI-QDA-Standard mit der Seitenzahl 0. Zusätzlich werden in QDAcity die X- und Y-Koordinaten als Prozentsätze gespeichert, wobei der Ursprung des Koordinatensystems oben links liegt. Im REFI-QDA-Standard hingegen hat das Koordinatensystem den Ursprung unten links, und die Koordinaten werden in Pixeln angegeben, wobei die tatsächliche Größe des PDF-Dokuments verwendet wird.

Aus diesem Grund müssen die Werte von QDAcity zunächst umgerechnet werden. Die Seitenzahl wird einfach um 1 verringert, um sie an den REFI-QDA-Standard anzupassen. Für die Koordinaten muss die Dimension des PDF-Dokuments berücksichtigt werden. Diese kann durch das Erstellen eines PDF-Dokuments in JavaScript und das Befüllen mit dem Inhalt des Dokuments ermittelt werden. Dadurch lassen sich die Dimensionen des Dokuments, die Anzahl der Seiten sowie der Textkörper ermitteln.

Die Dimensionen der Seiten werden ermittelt, indem man jede Seite des PDF-Dokuments in Originalgröße betrachtet und die Breite sowie die Höhe abruft. Diese Werte werden für jede Seite gespeichert, da die Seiten eines PDF-Dokuments unterschiedliche Größen aufweisen können. Die Dimensionen jeder Seite werden in einem Array gespeichert, wobei der Index des Arrays der Seitenzahl entspricht. Nachdem die Dimensionen aller Seiten des PDF-Dokuments ermittelt wurden, können die prozentualen Koordinaten aus QDAcity in die von REFI-QDA geforderten Koordinaten umgerechnet werden. Die Umrechnung erfolgt folgendermaßen:

- **X-Koordinate:** Die prozentuale X-Koordinate wird mit der Breite der jeweiligen Seite multipliziert, um die tatsächliche X-Position in Pixeln zu erhalten.
- **Y-Koordinate:** Da im REFI-QDA-Standard der Ursprung des Koordinatensystems unten links liegt (während in QDAcity der Ursprung oben links ist), muss die prozentuale Y-Koordinate zuerst von 1 abgezogen werden (also 1 - Y-Koordinate). Danach wird das Ergebnis mit der Höhe der Seite multipliziert, um die tatsächliche Y-Position in Pixeln zu berechnen.

Diese Umrechnung gewährleistet, dass die Koordinaten korrekt auf die Dimensionen des PDF-Dokuments angewendet werden und den Anforderungen des REFI-QDA-Standards entsprechen.

Das Extrahieren des Textkörpers aus einem PDF-Dokument ist zwar technisch nicht besonders komplex, jedoch muss der extrahierte Text korrekt formatiert und konkateniert werden, um der originalen Struktur des Dokuments zu entsprechen. Der Textkörper wird dabei als Baumstruktur repräsentiert, bei der die obersten Knoten die Seiten des Dokuments darstellen, während die darunter liegenden Knoten den eigentlichen Text repräsentieren. Jeder Knoten ist durchnummeriert, was die Reihenfolge der Textteile auf den jeweiligen Seiten wiedergibt. Diese Textknoten enthalten neben dem Text selbst auch einige nützliche Attribute, wie etwa die Textgröße und die Koordinaten des Textes auf der Seite. Diese Attribute sind entscheidend für die korrekte Formatierung und Verknüpfung der Textknoten.

Im REFI-QDA-Standard gibt es keine spezifischen Vorgaben für die Struktur der Textrepräsentation eines PDF-Dokuments. Daher wurde entschieden, die allgemeine Struktur eines PDF-Dokuments als Grundlage für die Textrepräsentation zu verwenden. Das bedeutet, dass im extrahierten Text jede neue Zeile, die im PDF-Dokument beginnt, auch als neue Zeile im Textkörper umgesetzt wird.

Ein weiteres Detail ist, dass in den Textknoten selbst keine expliziten Zeilenumbrüche enthalten sind. Stattdessen wird diese Information durch die Differenz der Y-Koordinaten zwischen benachbarten Textknoten abgeleitet. Wenn die Differenz der Y-Koordinaten größer als die Höhe des Textes (d. h. die Textgröße) ist, wird dies als Beginn einer neuen Zeile interpretiert.

Die `PlainTextSelection` benötigt lediglich die Anfangs- und Endposition der Codierung. QDAcity speichert diese Informationen als Punkte im Baum, der die Struktur des extrahierten Textes repräsentiert. Für jede Codierung gibt QDAcity die Seitenzahl, die Nummer des Textknotens und die Position innerhalb des Textknotens an.

Ein wichtiger Aspekt dabei ist, dass QDAcity Textknoten, die nur ein Leerzeichen enthalten, nicht als Knoten zählt. Das bedeutet, dass die von QDAcity angegebene Nummer für den Textknoten nicht die tatsächliche Nummer im Baum widerspiegelt. Um die richtige Position zu ermitteln, muss man selbst die Zählung vornehmen und dabei die Knoten, die nur Leerzeichen enthalten, überspringen, um den richtigen Textknoten zu finden.

Verarbeitung von Textdokumenten
Der Inhalt von Textdokumenten wird ebenfalls aus dem GCS abgerufen. Im Unterschied zu PDF-Dokumenten wird der Textinhalt jedoch aus einer Ydoc-Datei extrahiert. Da es sich hierbei um eine binäre Datei handelt, muss der Inhalt zunächst in einen Yjs-Editor geladen werden, um darauf zugreifen und ihn auslesen zu können. In diesem Editor wird der Text, ähnlich wie im Fall des PDF-Dokuments, in einer Baumstruktur repräsentiert. Jedoch stellt jeder Knoten eine einzelne Textzeile dar. Auf diese Weise kann durch die Baumstruktur iteriert und die Textzeilen unter Berücksichtigung von Zeilenumbrüchen miteinander verknüpft werden. Der daraus resultierende Text wird anschließend als Textdokument im `source`-Ordner des QDPX-Archivs abgelegt.

Durch `PlainTextSelection` werden die Codierungen in Textdokumenten ebenfalls repräsentiert. Dafür sind, wie auch bei den PDF-Dokumenten, die Anfangs- und Endposition der Codierung erforderlich. QDAcity speichert diese Informationen als Zeiger im Ydoc. Beim Abrufen dieser Informationen erhält man die Zeile sowie die genaue Position innerhalb der Zeile für die Anfangs- und Endmarkierungen der Codierung. Zur Bestimmung der exakten Position muss daher lediglich die Anzahl der Zeichen im Textinhalt bis zu den jeweiligen Positionen gezählt werden.

Archiv-Export
Nachdem die Platzhalterwerte in den `PDFSelection`- und `PlainTextSelection`-Objekten mit den korrekten Werten gefüllt wurden, wird der XML-String in eine Datei namens `project.qde` umgewandelt und in das QDPX-Archiv eingefügt. Dieses Archiv wird anschließend mithilfe einer signierten URL dem Nutzer zum Download zur Verfügung gestellt.

4.3 Import

Im Gegensatz zum Exportprozess erfolgt der Import weiterhin im Frontend. Dies liegt daran, dass die hierfür erforderlichen Funktionen bereits im Frontend vorhanden sind und die Bearbeitung der Dokumenteninhalte ausschließlich im Frontend durchgeführt werden kann.

Der Importprozess wird durch einen Klick des Nutzers auf den Import-Button initiiert. Daraufhin öffnet sich ein Dateiauswahl-Dialog, in dem die gewünschte Datei für den Import ausgewählt werden kann. Für jeden Importtyp wird überprüft, ob die ausgewählte Datei das vorgegebene Dateiformat erfüllt. Nach einem erfolgreichen Upload beginnt die Verarbeitung der Datei. Während dieses Prozesses wird dem Nutzer eine Fortschrittsanzeige präsentiert, deren Länge von der Anzahl der zu verarbeitenden Elemente abhängt. Beispielsweise basiert die Anzeige beim Import eines Codebuchs auf der Anzahl der enthaltenen Codes, während sie bei einem Projekt-Import die Anzahl der Codes, Codierungen und Dokumente berücksichtigt.

Um einen Code zu erstellen, werden lediglich der Name und der übergeordnete Code benötigt. Der Code wird anschließend im Ydoc des Projekts gespeichert. Das Code-Objekt wird zurückgegeben, sodass auch die Attribute des Codes später bearbeitet werden können.

4.3.1 Codebuch-Import

Im Zuge der Erweiterung der Exportmöglichkeiten für Codebücher, die nun sowohl im CSV- als auch im QDC-Format bereitgestellt werden, erfolgt auch der Import beider Formate.

4.3.1.1 CSV-Codebuch-Import

Wie in Kapitel 4.2.2.1 beschrieben, enthält das CSV-Codebuch spezifische Daten. Beim Importprozess wird geprüft, ob die CSV-Datei die erforderlichen Spaltennamen im Header aufweist. Die Reihenfolge der Spalten ist dabei irrelevant, um potenzielle zukünftige Änderungen am Exportformat zu berücksichtigen. Entscheidend ist lediglich, dass alle notwendigen Spalten für die Verarbeitung vorhanden sind.

Nach der Prüfung des Headers wird die CSV-Datei zeilenweise verarbeitet, wobei jede Zeile einen einzelnen Code repräsentiert. Da es sich um ein von QDAcity entwickeltes Format handelt, sollte die erste Zeile nach dem Header den Code „Code System“ enthalten. Dieser Code wird jedoch nicht neu erstellt, da jedes Projekt in QDAcity diesen als Wurzelknoten des Codebuchs verwendet. Um sicherzustellen, dass andere QDA-Software, die dieses Format möglicherweise kopiert, korrekt behandelt wird, wird der „Code System“-Code wie folgt überprüft: Für jede Zeile wird die Länge des Code-Pfads analysiert. Falls die Länge des Pfads 1 beträgt, handelt es sich um einen Wurzelknoten. In diesem Fall wird der Name des Codes überprüft. Ist der Name „Code System“, wird die Erstellung des Codes übersprungen. Stattdessen wird das vorhandene Code-Objekt für „Code System“ abgerufen und für die weitere Verarbeitung der CSV-Datei genutzt.

Für alle anderen Codes läuft der Prozess wie folgt ab: Zunächst werden alle direkten Nachfolger des aktuellen Codes ermittelt. Dies geschieht durch einen Vergleich der Code-Pfade. Codes, deren Pfad den des aktuellen Codes enthält und die um genau eine Ebene länger sind, werden in einer Liste gesammelt. Anschließend wird ein neues Code-Objekt für den aktuellen Code erstellt, wobei die Attribute aus der CSV-Zeile ausgelesen und gesetzt werden. Nach der erfolgreichen Erstellung des Codes im Projekt wird die Liste der direkten Nachfolger durchlaufen. Jeder Nachfolger wird dem aktuellen Code als Vorgänger zugewiesen, der Ablauf wird rekursiv für jeden Nachfolger wiederholt. Dieser Prozess setzt sich fort, bis keine Nachfolger-Codes mehr existieren. Dadurch wird jedem Code der korrekte Vorgänger zugeordnet, ohne dass Zwischenspeicherungen erforderlich sind.

Im CSV-Export wurde die Code-ID neu hinzugefügt, um den Importprozess zu optimieren. Diese ID ermöglicht es, bereits existierende Codes zu identifizieren und deren erneute Erstellung zu vermeiden. Während des Imports wird die Code-ID überprüft. Existiert ein Code-Objekt mit der entsprechenden ID, wird dies nicht neu erstellt. Stattdessen wird das bestehende Code-Objekt abgerufen und für die weitere Verarbeitung der übrigen Codes verwendet.

4.3.1.2 QDC-Codebuch-Import

Der QDC-Codebuch-Import funktioniert im Wesentlichen genauso wie der CSV-Codebuch-Import. Zunächst wird die XML-Datei an einen XML-Parser übergeben, der überprüft, ob das Format korrekt ist. Wenn dies der Fall ist, wird das <Codebook>-Element extrahiert, das die Codes enthält. Diese Codes werden dann weiterverarbeitet und hinzugefügt. Der Ablauf erfolgt rekursiv, ähnlich wie beim CSV-Codebuch-Import. Der Hauptunterschied liegt darin, dass eine andere Methode verwendet wird, um die direkten Nachfolger der Codes zu ermitteln, und dass im QDC-Codebuch zusätzliche Attribute hinterlegt sind.

Der allgemeine Ablauf bleibt folgendermaßen: Zu Beginn wird der Wurzelknoten des XML-Dokuments verarbeitet und alle direkten Nachfolger-Codes werden mithilfe der querySelectorAll-Funktion ermittelt. Die querySelectorAll-Funktion ist eine Methode, die in einer XML-Struktur bestimmte Elemente basierend auf einem CSS-ähnlichen Selektor auslesen kann. Nachdem die Nachfolger-Codes ermittelt wurden, wird überprüft, ob der Wurzelknoten dem „Code System“-Code entspricht, da QDAcity diesen immer als Wurzelknoten verwendet. Ist dies der Fall, wird die Erstellung des Codes übersprungen und der „Code System“-Code wird extrahiert und zur weiteren Bearbeitung weitergegeben.

Falls der Wurzelknoten nicht dem `Code System`-Code entspricht, wird er trotzdem extrahiert und als Vorgänger des Wurzelknotens gesetzt, da in QDAcity nur dieser als Wurzelknoten existieren kann. Aus dem <Code>-Element des Wurzelknotens wird der Name des Codes ausgelesen und zusammen mit dem `Code System`-Code ein neues Code-Objekt erstellt. Anschließend werden die Farbe und die Beschreibung aus dem <Code>-Element des Wurzelknotens ausgelesen und im neuen Code-Objekt gesetzt.

Anschließend wird über die Liste der direkten Nachfolger-Codes iteriert und der Vorgang wird mit dem neu erstellten Code-Objekt als Vorgänger wiederholt. Dieser Prozess wird so lange fortgesetzt, bis keine Nachfolger mehr vorhanden sind.

4.3.2 Projekt-Import

Der QDPX-Projekt-Import ähnelt dem QDC-Codebuch-Import, jedoch werden beim Projekt-Import mehr Daten verarbeitet. Zunächst wird die QDE-Datei aus dem QDPX-Archiv abgerufen. Diese QDE-Datei wird für den Import abgearbeitet. Wie beim QDC-Codebuch-Import wird die QDE-Datei einem XML-Parser übergeben. Zuerst wird das Codebuch bearbeitet, gefolgt von den Dokumenten und Codierungen. Da das Codebuch bereits im QDC-Codebuch behandelt wurde, wird dieser Schritt hier ausgelassen. Der einzige Unterschied liegt darin, dass die Code-Objekte mit ihren GUIDs aus den <Code>-Elementen gespeichert werden, damit später die Codierungen diese aufrufen und den richtigen Code zuordnen können.

Verarbeitung von PDF-Dokumenten
Für die Dokumente wird das <Sources>-Element abgerufen, das alle Dokumente sowie deren Codierungen enthält. Aus dem <Sources>-Element werden nur die <PDFSource>- und <TextSource>-Elemente extrahiert, da QDAcity nur PDF- und Text-Dokumente unterstützt. Diese Elemente werden einzeln bearbeitet.

Beim <PDFSource>-Element wird zunächst das zugehörige PDF-Dokument aus dem `source`-Ordner abgerufen und mit der vorhandenen Upload-Funktion in den GCS hochgeladen. Diese Funktion erstellt gleichzeitig einen Eintrag in der Datenbank, indem sie die entsprechende Backend-Funktion aufruft. Nachdem das PDF-Dokument hochgeladen wurde und ein Eintrag in der Datenbank erstellt wurde, wird ein Dokument-Objekt zurückgegeben, das das PDF-Dokument repräsentiert. Mit diesem Dokument-Objekt können, wie beim Projekt-Export, die Dimensionen sowie der Textinhalt des PDF-Dokuments abgerufen werden, die für die Erstellung der Codierungen erforderlich sind.

Im PDF-Dokument gibt es zwei Arten von Codierungen: PDFAreaCoding und PDFTextCoding. Nachdem das PDF-Dokument hochgeladen wurde, werden alle <PDFSelection>-Elemente abgerufen, um daraus die PDFAreaCoding zu erstellen. Für die Erstellung einer PDFAreaCoding werden die folgenden Daten benötigt:

- **Coding-ID:** Wird automatisch generiert.
- **Codierungstyp:** Wird je nach Art der Codierung gesetzt.
- **Autor:** Wird auf den aktuellen Nutzer gesetzt.
- **localCodeID:** ID des angewandten Codes.
- **documentID:** ID des Dokuments, in dem sich die Codierung befindet.
- **Preview:** Eine Vorschau, falls vorhanden.
- **Koordinaten der Selektion:** Dies bezieht sich auf die genaue Position der Selektion innerhalb des PDF-Dokuments, basierend auf den Koordinaten im Dokument.

Bei PDFTextCoding werden dieselben Daten benötigt, jedoch wird anstelle der Koordinaten die Position im Text verwendet.

Der Importprozess erfolgt im Wesentlichen umgekehrt zum Export. Wie in Kapitel 4.2.5 beschrieben, speichert QDAcity die X- und Y-Koordinaten als Prozentwerte relativ zum Ursprung des Koordinatensystems (oben links), während im REFI-QDA-Standard die Koordinaten in Pixeln angegeben werden, wobei der Ursprung unten links liegt. Daher ist eine Umrechnung erforderlich, bei der die Dimensionen der PDF-Seiten berücksichtigt werden. Nachdem die Dimension des PDF-Dokuments ermittelt wurde, werden die Koordinaten aus dem <PDFSelection>-Element extrahiert und entsprechend umgerechnet.

Die X-Koordinate wird durch die Breite des Dokuments geteilt, während die Y-Koordinate durch die Höhe des Dokuments geteilt und anschließend von 1 abgezogen wird. Die Seitenzahl wird um 1 erhöht, da QDAcity die Seitenzählung bei 1 beginnen lässt, im Gegensatz zum REFI-QDA-Standard, der bei 0 startet.

Für PDFTextCoding wird die Position im Textbaum gespeichert. Da QDAcity Textknoten mit nur einem Leerzeichen nicht berücksichtigt, muss zunächst der Textbaum durchlaufen und die Textknoten mit Leerzeichen gezählt werden, um diese von den tatsächlichen Textpositionen abzuziehen.

Die erstellten Codierungen werden in einem Array gesammelt, das am Ende im Projekt-Ydoc gespeichert wird.

Verarbeitung von Textdokumenten
Für das <TextSource>-Element wird zunächst ein Dokument-Objekt erstellt und in der Datenbank registriert. Anschließend wird auf das Dokument-Ydoc zugegriffen, in dem der Text des Textdokuments gespeichert wird. Dies ist notwendig, da bei Textdokumenten die Anfangs- und Endpositionen der Codierungen relativ zur Position im Textdokument-Ydoc gespeichert werden. Diese Speicherung ist erforderlich, da Textdokumente editiert werden können, was zu Änderungen der Textpositionen führt. Daher wird der Punkt im Dokument-Ydoc als „Zeiger“ gespeichert, der sich mitbewegt, wenn der Text im Dokument geändert wird. Abgesehen davon sind die Attribute der Codierung mit denen des PDF-Dokuments identisch.

Nachdem das Dokument-Ydoc mit dem Text des Textdokuments befüllt wurde und die Textcodierungen erstellt sind, wird das Dokument-Ydoc in den GCS hochgeladen, um es zu aktualisieren. Die Textcodierungen werden gemeinsam mit den anderen Codierungen gesammelt.

Nachdem alle Dokumente hochgeladen und alle Codierungsobjekte erstellt wurden, werden die Codierungen im Projekt-Ydoc gespeichert, womit der Importprozess abgeschlossen ist.

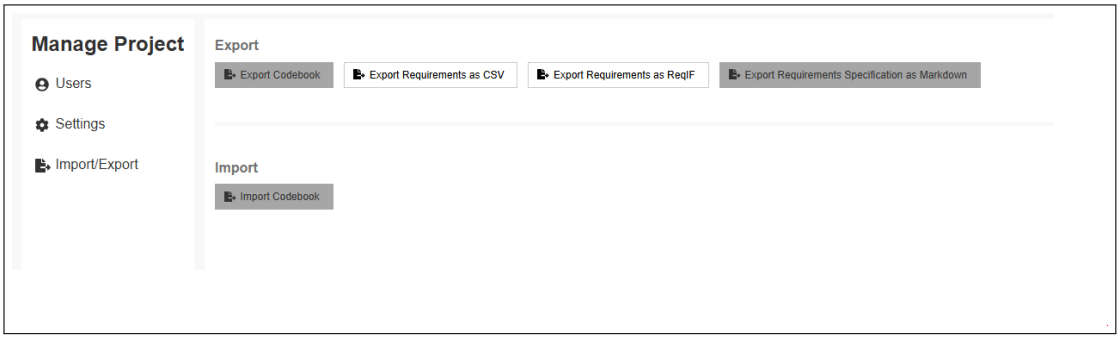

Abbildung 4.6: Zustand der Import-Export-Seite vor Beginn der Arbeit

4.4 Benutzeroberfläche

Neben der Funktionserweiterung um den Import und Export wurde die Benutzeroberfläche der Import-Export-Seite neugestaltet, um diese für die Nutzer anschaulicher und verständlicher zu machen. Dabei wurden Beschreibungen zu den jeweiligen Import- und Exportfunktionen hinzugefügt sowie neue UI-Elemente integriert, die den Fortschritt der jeweiligen Prozesse visualisieren.

Zunächst wurde eine Button-Komponente eingeführt, die für alle Importe und Exporte verwendet wird. Diesem Button werden folgende Parameter übergeben:

- **onClick:** Funktion, die ausgeführt wird, wenn auf den Button geklickt wird.
- **id:** Eindeutiger Identifier zur Unterscheidung der Button-Instanzen.
- **label:** Text, der auf dem Button angezeigt wird.
- **tooltip:** Text, der erscheint, wenn der Mauszeiger über den Button bewegt wird.
- **isLoading:** Status, der anzeigt, ob gerade eine Aktion ausgeführt wird.

Dadurch kann der Button individuell angepasst werden. Die Eigenschaft `isLoading` steuert die Anzeige des Buttons: Ist der Wert `true`, wird eine Ladeanimation mit dem Text „Loading...“ angezeigt. Ist der Wert `false`, erscheint der Button in seiner normalen Form mit dem definierten Text. Diese Funktionalität soll dem Nutzer, insbesondere beim Export, verdeutlichen, dass im Hintergrund ein Prozess ausgeführt wird, um den Export bereitzustellen.

Abbildung 4.8 zeigt die allgemeine Struktur der Frontend-Komponenten, die für die Darstellung und Funktionalität der Import- und Export-Seite zuständig sind. Es werden beispielhaft nicht alle Import- und Export-Typen dargestellt.

Die Hauptkomponente `Import/Export` stellt die Import-Export-Seite dar. Sie integriert die `Import`- und `Export`-Komponenten, welche widerum Unterkomponen-

ten wie `Header`, `Beschreibung` und `Button` enthalten. Die Button-Komponenten initiieren schließlich die Funktionalität, indem sie die jeweils zuständigen Funktionskomponenten aufrufen.

Theoretisch wäre es möglich, die Komponenten `Header`, `Beschreibung` und `Button` zu einer einzigen Komponente zusammenzufassen und diese dann mit den entsprechenden Inhalten an die `Import`- und `Export`-Komponenten zu übergeben. Allerdings bietet die aktuelle Struktur den Vorteil, dass jede Komponente eine klar definierte Funktion hat, was zukünftige Anpassungen erleichtert. So kann die `Header`-Komponente beispielsweise mehrere Import- oder Export-Funktionen unterstützen, da für den Codebuch-Import verschiedene Varianten existieren. Würde man alle Komponenten zusammenfassen, müsste man sicherstellen, dass nur eine Instanz des Headers existiert, die für alle zugehörigen Funktionen gilt. Dies könnte die Flexibilität und Wartbarkeit erschweren.

Während beim Export eine Ladeanimation verwendet wird, um dem Nutzer anzuzeigen, dass im Hintergrund ein Prozess läuft, zeigt der Import eine Fortschrittsanzeige, die die verbleibende Anzahl der noch zu verarbeitenden Daten darstellt.

Für den CSV-Coding-Summary-Export gibt es einen zusätzlichen Button, mit dem der Nutzer auswählen kann, welche Variante des Exports bei einem Klick ausgeführt werden soll.

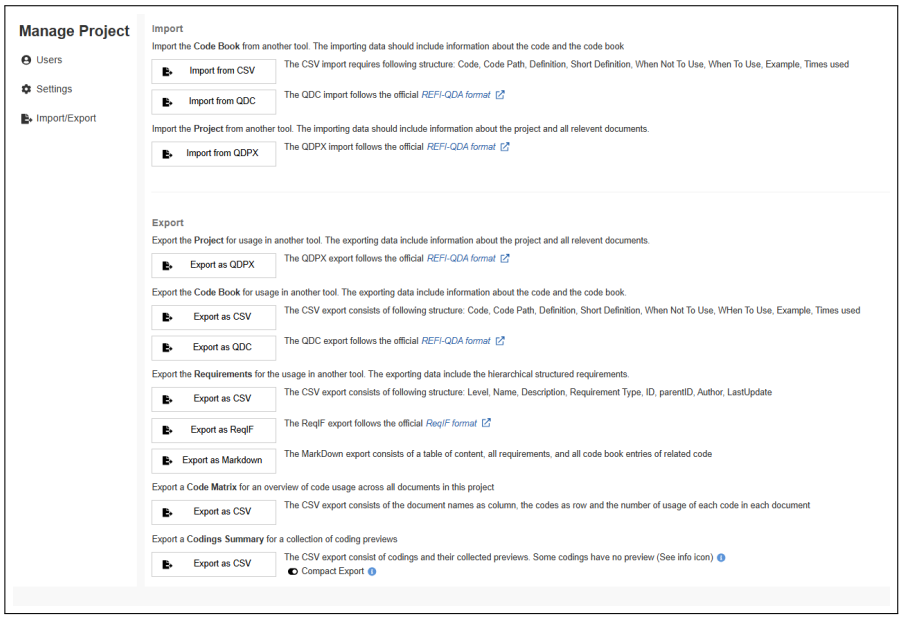

Abbildung 4.7: Aktueller Stand der Import-Export-Seite nach der Überarbeitung

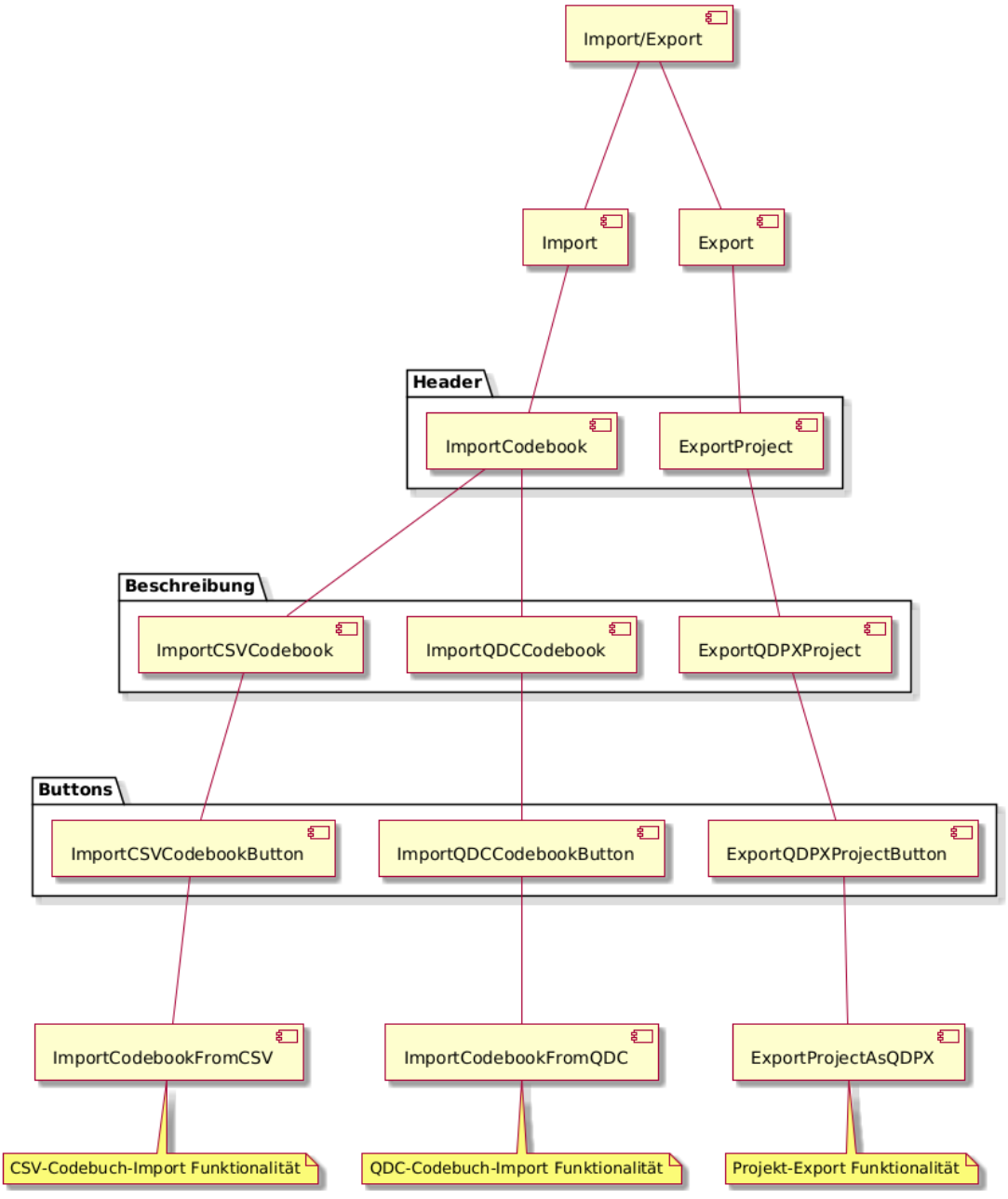

Abbildung 4.8: Übersicht der Komponenten im Frontend

5 Evaluation

5.1 Anforderungsevaluation

5.1.1 Funktionale Anforderungen

FA-01: *Der Nutzer kann ein Projekt aus QDAcity exportieren.*

FA-02: *Der Nutzer kann ein Projekt in QDAcity importieren.*

Durch die Einführung des REFI-QDA-Standards ist es nun möglich, Projekte sowohl zu exportieren als auch zu importieren.

FA-01 und FA-02 erfüllt

FA-03: *Der Nutzer kann ein Projekt aus QDAcity in eine andere QDA-Software exportieren.*

FA-04: *Der Nutzer kann ein Projekt aus einer anderen QDA-Software in QDAcity importieren.*

Die Einführung des REFI-QDA-Standards ermöglicht nun die Umsetzung dieser Funktionalitäten. Eine detaillierte Evaluation hierzu findet sich in Kapitel 5.2.

FA-03 und FA-04 erfüllt

FA-05: *Der Nutzer kann das Projekt lokal auf seinem Computer im schreibgeschützten Modus betrachten.*

Als Motivation dieser Arbeit wurde die Idee verfolgt, eine schreibgeschützte Version des Coding-Editors bereitzustellen. Ziel war es, die Möglichkeit zu schaffen, ein Projekt herunterzuladen und lokal auf einem Rechner betrachten zu können. Dies hätte es den Nutzenden ermöglicht, die Projektdaten und Codierungen ohne Bearbeitungsrechte einzusehen.

Geplant war, den Coding-Editor mithilfe von HTML zu visualisieren und die Projektdaten in einer geeigneten Form darzustellen. Bedauerlicherweise konnte im Rahmen dieser Arbeit keine abschließende Lösung für die Umsetzung dieser Idee gefunden werden. Weitere Untersuchungen könnten sich darauf konzentrieren,

die Daten des Projekts in einem portablen und leicht zugänglichen Format zu exportieren, um eine lokale Einsichtnahme zu ermöglichen.

FA-05 nicht erfüllt

FA-06: *Der Nutzer kann ein Codebuch aus QDAcity exportieren.*

FA-07: *Der Nutzer kann ein Codebuch in QDAcity importieren.*

Durch die Einführung des QDC-Formats stehen nun zwei Formate für den Austausch von Codebüchern zur Verfügung, was den Import und Export flexibler gestaltet.

FA-06 und FA-07 erfüllt

FA-08: *Der Nutzer kann ein Codebuch aus QDAcity in eine andere QDA-Software exportieren.*

FA-09: *Der Nutzer kann ein Codebuch aus einer anderen QDA-Software in QDAcity importieren.*

Die Einführung des REFI-QDA-Standards ermöglicht nun die Umsetzung dieser Funktionalitäten. Eine detaillierte Evaluation hierzu findet sich in Kapitel 5.2.

FA-08 und FA-09 erfüllt

FA-10: *Der Nutzer kann eine Übersicht aller Codierungen exportieren.*

Durch die Einführung des CSV-Coding-Summarys wird eine Übersicht aller Codierungen bereitgestellt, die sowohl eine Vorschau der jeweiligen Codierungen als auch den zugehörigen Code-Pfad enthält. Es stehen zwei Varianten zur Verfügung: Die erste Variante exportiert jede Codierung einzeln, während die zweite die Codierungen pro Code bündelt und zusammengefasst ausgibt.

FA-10 erfüllt

FA-11: *Der Nutzer kann eine Übersicht über die Anzahl aller Codierungen pro Dokument exportieren.*

Die Einführung der CSV-Code-Matrix ermöglicht die Darstellung der Codierungsfrequenzen. In dieser Matrix wird die Anzahl der Codierungen pro Code und pro Dokument übersichtlich aufbereitet und exportiert.

FA-11 erfüllt

5.1.2 Nicht-funktionale Anforderungen

NFA-01: *Die Daten sollen mit verschiedenen QDA-Softwarelösungen ausgetauscht werden können, wobei mindestens zwei interoperable Systeme unterstützt werden müssen. (Interoperabilität)*

Die Einführung des REFI-QDA-Standards ermöglicht nun die Umsetzung dieser Funktionalitäten. Eine detaillierte Evaluation hierzu findet sich in Kapitel 5.2.

NFA-01 erfüllt

NFA-02: *Nur autorisierte Nutzer dürfen Projekte exportieren. (Sicherheit)*

Beim Export eines Projekts wird überprüft, ob der Nutzer die erforderlichen Berechtigungen für den Zugriff auf das Projekt besitzt. Erst nachdem diese Prüfung erfolgreich abgeschlossen wurde, wird der Exportprozess durchgeführt. Projekte sind grundsätzlich nur für berechtigte Nutzer zugänglich, sodass selbst dann, wenn Dritte die URL zum Projekt kennen, der Zugriff auf das Projekt verweigert wird, wenn sie keine entsprechenden Rechte besitzen. Dadurch wird sichergestellt, dass nur autorisierte Nutzer Projekte exportieren können, wodurch die Sicherheit und Vertraulichkeit der Projektdaten gewahrt bleibt.

NFA-02 erfüllt

NFA-03: *Der Import- und Exportprozess soll auch bei größeren Projekten mit vielen Dokumenten und Codierungen schnell und effizient ablaufen. Die Antwortzeiten der Anwendung während dieses Vorgangs sollten akzeptabel sein und sich im Bereich von wenigen Sekunden bis Minuten bewegen. (Performance/Zeiteffizienz)*

Um die Zeiteffizienz der Import- und Exportprozesse zu testen, wurden drei verschiedene Projekte sowohl exportiert als auch importiert.

- **Projekt A:** 0 Dokumente, 1 Code, 0 Codierungen
- **Projekt B:** 5 Dokumente, 10 Codes, 10 Codierungen
- **Projekt C:** 10 Dokumente, 25 Codes, 50 Codierungen

Jeder Import- und Exportvorgang wurde mit jedem Projekt dreimal durchgeführt, und der Mittelwert der Ergebnisse wurde in den folgenden Tabellen dargestellt.

Projekt	CSV-Codebuch	QDC-Codebuch	QDPX-Projekt
A	6 ms	8 ms	11 ms
B	75 ms	75 ms	10802 ms
C	231 ms	203 ms	18227 ms

Tabelle 5.1: Dauer des Imports

Projekt	CSV-Codebuch	QDC-Codebuch	QDPX-Projekt
A	21 ms	18 ms	27 ms
B	348 ms	20 ms	4777 ms
C	366 ms	19 ms	8699 ms

Tabelle 5.2: Dauer des Exports

Projekt	Code-Matrix	Normale Coding Summary	Kompakte Coding Summary
A	20 ms	23 ms	20 ms
B	333 ms	368 ms	441 ms
C	315 ms	333 ms	324 ms

Tabelle 5.3: Dauer des Exports 2

Aus den Tabellen lässt sich entnehmen, dass die Zeiten für alle Import- und Exportprozesse vernachlässigbar sind, da sie alle unter 500 ms liegen. Einzig der Projekt-Import und -Export nehmen etwas mehr Zeit in Anspruch, wobei der Export schneller als der Import ist. Insgesamt erscheinen die Zeiten noch im akzeptablen Rahmen. Da jedoch keine genaue Einschätzung über die maximale Größe der Projekte vorliegt, lässt sich nicht abschließend beurteilen, ob dieses Zeitverhalten auch bei größeren Projekten noch angemessen ist. Es ist zu beachten, dass diese Zeiten auf einer lokalen Instanz von QDAcity gemessen wurden. Da es keine Möglichkeit gibt, die Zeiten in der tatsächlichen Anwendung zu messen, könnten bei deren Nutzung Abweichungen auftreten.

NFA-03 ist vermutlich erfüllt

NFA-04: *Die Benutzeroberfläche für den Import und Export von Projekten und Codebüchern muss intuitiv und leicht verständlich gestaltet sein, sodass Nutzer mit minimalem Aufwand die gewünschten Daten importieren oder exportieren können. (Benutzbarkeit)*

Im Zuge der Neugestaltung der Import-/Export-Seite wurden klare und prägnante Beschreibungen für jede Funktion hinzugefügt. Diese sollen dem Nutzer helfen, schnell zu verstehen, welche Funktion hinter jeder Option steckt, und den Nutzer so bei der Auswahl des richtigen Prozesses unterstützen. Durch diese Verbesserungen wird der Aufwand für die Durchführung von Import- und Exportvorgängen minimiert, was zu einer benutzerfreundlicheren und effizienteren Nutzung der Software führt.

NFA-04 erfüllt

NFA-05: *Bei längeren Import- oder Exportvorgängen soll der Nutzer über den Fortschritt informiert werden, um Unklarheiten zu vermeiden. (Transparenz)*

NFA-09: *Im Falle von Fehlern während des Import- oder Exportvorgangs soll der Nutzer klare Informationen darüber erhalten, was schiefgelaufen ist. (Fehlerbehandlung)*

Im Zuge der Überarbeitung der Import-/Export-Seite wurde für die Export-Buttons ein Ladebutton hinzugefügt, der dem Nutzer visuell signalisiert, dass der Exportvorgang gerade ausgeführt wird. Für den Importprozess wurde eine Fortschrittsanzeige implementiert, die anzeigt, wie viele Elemente noch importiert bzw. erstellt werden müssen. Sollte der Import oder Export fehlschlagen, wird eine entsprechende Fehlermeldung ausgegeben, um den Nutzer auf das Problem hinzuweisen.

NFA-05 und NFA-09 erfüllt

NFA-06: *Die Import- und Exportkomponenten sollen leicht wartbar und anpassbar sein, damit sie bei Bedarf schnell aktualisiert werden können. (Wartbarkeit)*

NFA-07: *Neue Import- oder Exportfunktionen sollen einfach integrierbar sein. (Erweiterbarkeit)*

Durch das Refactoring wurde für die Exportkomponente ein dedizierter Export-Endpoint im Java-Backend erstellt, der ausschließlich für den Export zuständig ist. Dieser Endpoint ruft die jeweiligen Converter-Klassen auf, um die benötigten Daten für den Export zu verarbeiten. Im Frontend wurden Komponenten entwickelt, deren einzige Aufgabe es ist, diese Backend-Funktionalität für den Export zu initiieren.

Für den Importprozess wurden ebenfalls separate Komponenten im Frontend erstellt, die ausschließlich für die Import-Funktionalität zuständig sind. Durch diese Trennung sind für jeden Import- und Exporttyp jeweils spezifische Klassen oder Komponenten verantwortlich, was die Wartung und Anpassung erheblich vereinfacht. So können Änderungen schnell umgesetzt werden: Falls neue Export-Typen hinzukommen, muss im Export-Endpoint lediglich eine neue Funktion hinzugefügt werden, die den entsprechenden Converter aufruft. Zudem muss im Frontend eine neue Komponente erstellt werden, die diese Funktion nutzt. Beim Import reicht es aus, eine neue Komponente für die Import-Funktionalität zu erstellen, um den Prozess anzupassen.

NFA-06 und NFA-07 erfüllt

NFA-08: *Die Import- und Exportfunktionalitäten müssen umfassend getestet werden, um sicherzustellen, dass ihre Funktionalität auch bei Änderungen an QDAcity erhalten bleibt. (Zuverlässigkeit)*

Da der Exportprozess im Backend ausgeführt wird, wurden Unit-Tests entwickelt, um die entsprechende API zu überprüfen und sicherzustellen, dass sie die richtigen Daten zurückliefert. Diese Tests überprüfen insbesondere, ob der Export korrekt durchgeführt wird und die erwarteten Datenformate generiert werden.

Für den Importprozess hingegen wurden Acceptance-Tests erstellt, die den gesamten Importvorgang innerhalb von QDAcity simulieren. Diese Tests stellen sicher, dass nach dem Import alle relevanten Dokumente, Codes und Codierungen im Coding-Editor korrekt angezeigt und verarbeitet werden. Dadurch wird gewährleistet, dass die Funktionalitäten auch nach Änderungen an QDAcity weiterhin zuverlässig und fehlerfrei arbeiten.

NFA-08 erfüllt

5.2 Datenaustausch zwischen QDA-Tools

Um die Interoperabilität mit anderen QDA-Softwarelösungen zu prüfen, wurden sowohl der Import als auch der Export für Codebuch und Projekt in jeder unterstützten Software getestet. Dies stellte sicher, dass die Daten korrekt zwischen den verschiedenen QDA-Tools ausgetauscht werden konnten.

5.2.1 QDAcity QDC-Codebuch-Export

Export nach	QDAcity	Atlas.ti	MAXQDA	f4
Code	✓	✓	✓	✓
Color	✓	✓	✓	✓
Description	✓	✓	✓	✓
isCodable	-	?	-	-

Tabelle 5.4: QDC-Codebuch-Export in andere QDA-Softwareanwendungen

Das aus QDAcity exportierte Codebuch im QDC-Format kann ohne Probleme in QDA-Softwarelösungen wie QDAcity, Atlas.ti, MAXQDA sowie f4 importiert werden. Bei diesem Import werden alle Codes, deren Farbe und Beschreibung korrekt übernommen. Da QDAcity, MAXQDA und f4 immer alle Codes für Codierungen verwenden, lässt sich nicht sicher sagen, ob die `isCodable`-Eigenschaft korrekt übernommen wird. Allerdings bedeutet dies auch, dass diese drei Softwarelösungen standardmäßig isCodable auf `true` setzen.

In Atlas.ti wird die `isCodable`-Eigenschaft jedoch übernommen. Wenn `isCodable` auf `false` gesetzt ist, kann der Code in Atlas.ti nicht mehr verwendet werden. Das Codesystem von Atlas.ti ist allerdings etwas anders strukturiert: Es unterstützt maximal zwei Ebenen, was bedeutet, dass ein Kindcode keinen weiteren Code als Untercode haben kann. Diese Einschränkung in der Struktur von Atlas.ti führt dazu, dass Codes in dieser Software immer nur zwei Ebenen tief sein können, im Gegensatz zu anderen Systemen, die eine hierarchische Struktur mit mehr als zwei Ebenen unterstützen.

Ein weiteres wichtiges Detail ist, dass beim Import von Codebüchern in Atlas.ti eine bestimmte Formatierung erforderlich ist. Fehlt ein spezifischer String am Anfang der Datei, wird diese nicht als gültige Datei für den Import anerkannt. Dies ist bei den anderen Softwarelösungen nicht der Fall. Der benötigte String lautet:

```
<?xml version="1.0" encoding="utf-8"?>
```

5.2.2 QDAcity QDC-Codebuch-Import

Import von	QDAcity	Atlas.ti	MAXQDA	f4
Code	✓	✓	✓	✓
Color	✓	✓	✓	✓
Description	✓	✓	✓	✓
isCodable	-	-	-	-

Tabelle 5.5: QDC-Codebuch-Import von QDA-Softwareanwendungen

Alle QDC-Codebücher von QDAcity, Atlas.ti, MAXQDA und f4 konnten in QDAcity importiert werden, wobei alle Informationen übernommen wurden.

Die `isCodable`-Eigenschaft konnte nicht überprüft werden, da in QDAcity alle Codes für Codierungen verwendet werden.

5.2.3 QDAcity QDPX-Projekt-Export

Export nach	QDAcity	Atlas.ti	MAXQDA	f4
Text-Dokument	✓	✓	✓	✓
PDF-Dokument	✓	✓	✓	✗
TextCodings	✓	✓	✓	✓
PDFTextCodings	✓	✗	✗	✗
PDFAreaCodings	✓	✓	✓	✗
Codebuch	✓	?	✓	✓

Tabelle 5.6: QDPX-Projekt-Export in andere QDA-Softwareanwendungen

Ein aus QDAcity exportiertes Projekt kann problemlos wieder in QDAcity importiert werden. Da f4 keine PDF-Dokumente unterstützt, gehen diese beim Import verloren, ansonsten wird alles erfolgreich importiert. In MAXQDA wird alles bis auf die PDF-Text-Codierungen importiert. Es ist unklar, ob dies an der fehlenden Textrepräsentation für PDF-Dokumente in MAXQDA liegt oder ob es sich um ein ähnliches Problem wie bei Atlas.ti handelt. Auch Atlas.ti importiert alles außer den PDF-Text-Codierungen. Da die Textrepräsentation in Atlas.ti anders ist als in QDAcity, kann es sein, dass diese Codierungen beim Import verloren gehen. Zudem unterstützt Atlas.ti nur eine Codehierarchie mit maximal zwei Ebenen, was zu Problemen beim Import nicht verwendeter Codes führen kann.

5.2.4 QDAcity QDPX-Projekt-Import

Import von	QDAcity	Atlas.ti	MAXQDA	f4
Text-Dokument	✓	✓	✓	✓
PDF-Dokument	✓	✓	✓	-
TextCodings	✓	✓	✓	✓
PDFTextCodings	✓	?	(✓)	-
PDFAreaCodings	✓	✓	✓	-
Codebuch	✓	✓	✓	✓

Tabelle 5.7: QDPX-Projekt-Import von QDA-Softwareanwendungen

Der Import eines QDAcity-Projekts sowie eines aus f4 exportierten Projekts erfolgt problemlos. Auch Atlas.ti und MAXQDA funktionieren grundsätzlich einwandfrei, abgesehen von den PDF-Text-Codierungen. In MAXQDA werden diese als Flächencodierungen exportiert, die problemlos in QDAcity übernommen werden. Bei Atlas.ti hingegen werden die PDF-Text-Codierungen importiert, jedoch verschiebt sich die Codierung aufgrund der unterschiedlichen Textrepräsentation des PDF-Dokuments um einige Zeichen. Leider konnte kein konsistentes Muster in der Textrepräsentation von Atlas.ti identifiziert werden, und der REFI-QDA-Standard enthält keine spezifischen Vorgaben zur Strukturierung der Textrepräsentation von PDF-Dokumenten. Auch auf Nachfrage bei den Verantwortlichen des REFI-QDA-Standards wurden keine detaillierten Informationen zur genauen Implementierung der Textrepräsentation für PDF-Dokumente bereitgestellt.

5.3 Möglichkeiten für Verbesserung

Obwohl die Erweiterung der Import- und Exportfunktionalität grundsätzlich erfolgreich umgesetzt wurde, gibt es weiterhin einige kleinere Probleme, die möglicherweise noch behoben werden müssen.

Ein Problem betrifft die Struktur bestimmter Elemente in QDAcity. Zum Beispiel wurde eine Ordnerstruktur eingeführt, um Dokumente zu kategorisieren. Diese Struktur wird beim Export eines Projekts derzeit jedoch nicht übernommen. Da der REFI-QDA-Standard keine Hierarchie im `source`-Ordner des QDPX-Archivs zulässt, müsste die Ordnerstruktur alternativ durch Sets abgebildet werden. Beim Import müsste dann überprüft werden, ob diese Sets korrekt zur Wiederherstellung der Ordnerstruktur verwendet werden.

Ein weiteres Thema betrifft die Beschreibung von Codes. Aktuell gibt es keine eindeutige Beschreibung für Codes, sondern lediglich ein Textfeld für eine Definition sowie eine Angabe dazu, wann ein Code verwendet oder nicht verwendet werden soll. Es stellt sich die Frage, ob diese Beschreibungen beim Export berücksichtigt werden sollen. Wenn ein QDAcity-Projekt erneut in QDAcity importiert wird, sollte dann die Struktur der Beschreibungen wiederhergestellt werden? Zurzeit werden Beschreibungen in der Definition abgelegt.

Ein weiteres Problem betrifft den Autor von Codierungen, der derzeit nicht exportiert wird, da der Autor für Codes in QDAcity leicht geändert werden kann. Dies scheint jedoch für Codierungen nicht der Fall zu sein. Allgemein könnte der Autor als weniger relevant angesehen werden. Auch wenn beim Projektimport die Nutzer angezeigt werden, ist es schwierig, die genaue Verbindung zum ursprünglichen Autor herzustellen.

Zusammenfassend lässt sich sagen, dass der REFI-QDA-Standard in QDAcity für den Austausch innerhalb von QDAcity angepasst werden kann. Es ist jedoch klar, dass beim Austausch von Projekten zwischen QDAcity und anderen QDA-Softwarelösungen Daten verloren gehen können. Die zentrale Frage bleibt, ob und inwieweit der Datenverlust innerhalb von QDAcity reduziert werden soll.

6 Fazit

Im Großen und Ganzen wurde das Ziel erreicht. Durch die Einführung des REFI-QDA-Standards konnte der Datenaustausch zwischen QDAcity und anderen QDA-Softwarelösungen ermöglicht werden. Dies könnte dazu führen, dass Forscher QDAcity in Betracht ziehen, da es Funktionen bieten kann, die andere QDA-Softwarelösungen möglicherweise nicht haben. Vor der Einführung dieser Interoperabilität hätten Forscher aufgrund der fehlenden Möglichkeit, ihre Projekte und Ergebnisse zu migrieren, wahrscheinlich QDAcity nicht in Betracht gezogen.

Obwohl das Ziel erreicht wurde, gibt es weiterhin Potenzial für Optimierungen der Import-/Export-Funktionen. Das übergeordnete Ziel sollte sein, den Datenverlust während des Exports und Imports so gering wie möglich zu halten.

Neben dem Datenaustausch zwischen QDA-Softwareanwendungen wurden auch zwei Exportformate eingeführt, die die Nutzer von QDAcity bei der Weiterverarbeitung ihrer Analyse unterstützen sollen. Dies sollte ebenfalls berücksichtigt werden, jedoch erfordert die Entwicklung weiterer Formate das Feedback von Nutzern, um zu ermitteln, welche Formate sie benötigen, um ihre Ergebnisse effizient weiterzuverarbeiten.

Anhänge

A XML-Schema für den Austausch von Projektdaten im REFI-QDA-Standard

```
<xsd:schema xmlns="urn:QDA-XML:project:1.0" xmlns:xsd="http://www.w3.org/2001/
    ↪ XMLSchema" targetNamespace="urn:QDA-XML:project:1.0" elementFormDefault
    ↪ ="qualified" attributeFormDefault="unqualified" version="1.0">
    <!-- ===== Element Declarations ===== -->
    <xsd:element name="Project" type="ProjectType">
        <xsd:annotation>
            <xsd:documentation>This element MUST be conveyed as the root
                ↪ element in any instance document based on this Schema
                ↪ expression</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <!-- ===== Type Definitions ===== -->
    <xsd:complexType name="ProjectType">
        <xsd:sequence>
            <xsd:element name="Users" type="UsersType" minOccurs="0"/>
            <xsd:element name="CodeBook" type="CodeBookType" minOccurs="0"/>
            <xsd:element name="Variables" type="VariablesType" minOccurs="0"/>
            <xsd:element name="Cases" type="CasesType" minOccurs="0"/>
            <xsd:element name="Sources" type="SourcesType" minOccurs="0"/>
            <xsd:element name="Notes" type="NotesType" minOccurs="0"/>
            <xsd:element name="Links" type="LinksType" minOccurs="0"/>
            <xsd:element name="Sets" type="SetsType" minOccurs="0"/>
            <xsd:element name="Graphs" type="GraphsType" minOccurs="0"/>
            <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <!-- Note(s) that apply to the project as a whole -->
        </xsd:sequence>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="origin" type="xsd:string"/>
        <xsd:attribute name="creatingUserGUID" type="GUIDType"/>
        <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
        <xsd:attribute name="modifyingUserGUID" type="GUIDType"/>
        <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
        <xsd:attribute name="basePath" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="UsersType">
        <xsd:sequence>
            <xsd:element name="User" type="UserType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="UserType">
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
        <xsd:attribute name="name" type="xsd:string"/>
```

```
        <xsd:attribute name="id" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="CodeBookType">
        <xsd:sequence>
            <xsd:element name="Codes" type="CodesType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CodesType">
        <xsd:sequence>
            <xsd:element name="Code" type="CodeType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CodeType">
        <xsd:sequence>
            <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <xsd:element name="Code" type="CodeType" minOccurs="0" maxOccurs="
                ↪ unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="isCodable" type="xsd:boolean" use="required"/>
        <xsd:attribute name="color" type="RGBType"/>
    </xsd:complexType>
    <xsd:complexType name="CasesType">
        <xsd:sequence>
            <xsd:element name="Case" type="CaseType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CaseType">
        <xsd:sequence>
            <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="CodeRef" type="CodeRefType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <xsd:element name="VariableValue" type="VariableValueType"
                ↪ minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="SourceRef" type="SourceRefType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <xsd:element name="SelectionRef" type="SelectionRefType" minOccurs=
                ↪ "0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
        <xsd:attribute name="name" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="VariablesType">
        <xsd:sequence>
            <xsd:element name="Variable" type="VariableType" maxOccurs="
                ↪ unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
```

```
<xsd:complexType name="VariableType">
    <xsd:sequence>
        <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
    <xsd:attribute name="typeOfVariable" type="typeOfVariableType" use="
        ↪ required"/>
</xsd:complexType>
<xsd:complexType name="VariableValueType">
    <xsd:sequence>
        <xsd:element name="VariableRef" type="VariableRefType"/>
        <xsd:choice>
            <xsd:element name="TextValue" type="xsd:string" minOccurs="0"/>
            <xsd:element name="BooleanValue" type="xsd:boolean" minOccurs="0
                ↪ "/>
            <xsd:element name="IntegerValue" type="xsd:integer" minOccurs="0
                ↪ "/>
            <xsd:element name="FloatValue" type="xsd:decimal" minOccurs="0"/
                ↪ >
            <xsd:element name="DateValue" type="xsd:date" minOccurs="0"/>
            <xsd:element name="DateTimeValue" type="xsd:dateTime" minOccurs=
                ↪ "0"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SetsType">
    <xsd:sequence>
        <xsd:element name="Set" type="SetType" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="SetType">
    <xsd:sequence>
        <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
        <xsd:element name="MemberCode" type="CodeRefType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
        <xsd:element name="MemberSource" type="SourceRefType" minOccurs="0"
            ↪  maxOccurs="unbounded"/>
        <xsd:element name="MemberNote" type="NoteRefType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="name" type="xsd:string" use="required"/>
</xsd:complexType>
<xsd:complexType name="SourcesType">
    <xsd:choice maxOccurs="unbounded">
        <xsd:element name="TextSource" type="TextSourceType"/>
        <xsd:element name="PictureSource" type="PictureSourceType"/>
        <xsd:element name="PDFSource" type="PDFSourceType"/>
        <xsd:element name="AudioSource" type="AudioSourceType"/>
        <xsd:element name="VideoSource" type="VideoSourceType"/>
```

```
        </xsd:choice>
    </xsd:complexType>
    <xsd:complexType name="TextSourceType">
        <xsd:sequence>
            <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="PlainTextContent" type="xsd:string" minOccurs="0
                ↪ "/>
            <xsd:element name="PlainTextSelection" type="PlainTextSelectionType
                ↪ " minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="Coding" type="CodingType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <xsd:element name="VariableValue" type="VariableValueType"
                ↪ minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="richTextPath" type="xsd:string"/>
        <xsd:attribute name="plainTextPath" type="xsd:string"/>
        <xsd:attribute name="creatingUser" type="GUIDType"/>
        <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
        <xsd:attribute name="modifyingUser" type="GUIDType"/>
        <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
        <!-- Either PlainTextContent or plainTextPath MUST be filled, not both
            ↪ -->
    </xsd:complexType>
    <xsd:complexType name="PlainTextSelectionType">
        <xsd:sequence>
            <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="Coding" type="CodingType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="startPosition" type="xsd:integer" use="required"/>
        <xsd:attribute name="endPosition" type="xsd:integer" use="required"/>
        <xsd:attribute name="creatingUser" type="GUIDType"/>
        <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
        <xsd:attribute name="modifyingUser" type="GUIDType"/>
        <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
    </xsd:complexType>
    <xsd:complexType name="PictureSourceType">
        <xsd:sequence>
            <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="TextDescription" type="TextSourceType" minOccurs
                ↪ ="0"/>
            <xsd:element name="PictureSelection" type="PictureSelectionType"
                ↪ minOccurs="0" maxOccurs="unbounded"/>
```

```
        <xsd:element name="Coding" type="CodingType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
        <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
        <xsd:element name="VariableValue" type="VariableValueType"
            ↪ minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="path" type="xsd:string"/>
    <xsd:attribute name="currentPath" type="xsd:string"/>
    <xsd:attribute name="creatingUser" type="GUIDType"/>
    <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
    <xsd:attribute name="modifyingUser" type="GUIDType"/>
    <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
</xsd:complexType>
<xsd:complexType name="PictureSelectionType">
    <xsd:sequence>
        <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
        <xsd:element name="Coding" type="CodingType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
        <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="firstX" type="xsd:integer" use="required"/>
    <xsd:attribute name="firstY" type="xsd:integer" use="required"/>
    <xsd:attribute name="secondX" type="xsd:integer" use="required"/>
    <xsd:attribute name="secondY" type="xsd:integer" use="required"/>
    <xsd:attribute name="creatingUser" type="GUIDType"/>
    <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
    <xsd:attribute name="modifyingUser" type="GUIDType"/>
    <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
</xsd:complexType>
<xsd:complexType name="PDFSourceType">
    <xsd:sequence>
        <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
        <xsd:element name="PDFSelection" type="PDFSelectionType" minOccurs=
            ↪ "0" maxOccurs="unbounded"/>
        <xsd:element name="Representation" type="TextSourceType" minOccurs=
            ↪ "0"/>
        <xsd:element name="Coding" type="CodingType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
        <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
        <xsd:element name="VariableValue" type="VariableValueType"
            ↪ minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="name" type="xsd:string"/>
```

```
        <xsd:attribute name="path" type="xsd:string"/>
        <xsd:attribute name="currentPath" type="xsd:string"/>
        <xsd:attribute name="creatingUser" type="GUIDType"/>
        <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
        <xsd:attribute name="modifyingUser" type="GUIDType"/>
        <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
    </xsd:complexType>
    <xsd:complexType name="PDFSelectionType">
        <xsd:sequence>
            <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="Representation" type="TextSourceType" minOccurs=
                ↪ "0"/>
            <xsd:element name="Coding" type="CodingType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="page" type="xsd:integer" use="required"/>
        <xsd:attribute name="firstX" type="xsd:integer" use="required"/>
        <xsd:attribute name="firstY" type="xsd:integer" use="required"/>
        <xsd:attribute name="secondX" type="xsd:integer" use="required"/>
        <xsd:attribute name="secondY" type="xsd:integer" use="required"/>
        <xsd:attribute name="creatingUser" type="GUIDType"/>
        <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
        <xsd:attribute name="modifyingUser" type="GUIDType"/>
        <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
    </xsd:complexType>
    <xsd:complexType name="AudioSourceType">
        <xsd:sequence>
            <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="Transcript" type="TranscriptType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <xsd:element name="AudioSelection" type="AudioSelectionType"
                ↪ minOccurs="0" maxOccurs="unbounded"/>
            <xsd:element name="Coding" type="CodingType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <xsd:element name="VariableValue" type="VariableValueType"
                ↪ minOccurs="0" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="path" type="xsd:string"/>
        <xsd:attribute name="currentPath" type="xsd:string"/>
        <xsd:attribute name="creatingUser" type="GUIDType"/>
        <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
        <xsd:attribute name="modifyingUser" type="GUIDType"/>
        <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
```

```
</xsd:complexType>
<xsd:complexType name="AudioSelectionType">
    <xsd:sequence>
        <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
        <xsd:element name="Coding" type="CodingType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
        <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="begin" type="xsd:integer" use="required"/>
    <xsd:attribute name="end" type="xsd:integer" use="required"/>
    <xsd:attribute name="creatingUser" type="GUIDType"/>
    <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
    <xsd:attribute name="modifyingUser" type="GUIDType"/>
    <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
</xsd:complexType>
<xsd:complexType name="VideoSourceType">
    <xsd:sequence>
        <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
        <xsd:element name="Transcript" type="TranscriptType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
        <xsd:element name="VideoSelection" type="VideoSelectionType"
            ↪ minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element name="Coding" type="CodingType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
        <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
        <xsd:element name="VariableValue" type="VariableValueType"
            ↪ minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="path" type="xsd:string"/>
    <xsd:attribute name="currentPath" type="xsd:string"/>
    <xsd:attribute name="creatingUser" type="GUIDType"/>
    <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
    <xsd:attribute name="modifyingUser" type="GUIDType"/>
    <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
</xsd:complexType>
<xsd:complexType name="VideoSelectionType">
    <xsd:sequence>
        <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
        <xsd:element name="Coding" type="CodingType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
        <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="name" type="xsd:string"/>
```

```
        <xsd:attribute name="begin" type="xsd:integer" use="required"/>
        <xsd:attribute name="end" type="xsd:integer" use="required"/>
        <xsd:attribute name="creatingUser" type="GUIDType"/>
        <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
        <xsd:attribute name="modifyingUser" type="GUIDType"/>
        <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
    </xsd:complexType>
    <xsd:complexType name="TranscriptType">
        <xsd:sequence>
            <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="PlainTextContent" type="xsd:string" minOccurs="0
                ↪ "/>
            <xsd:element name="SyncPoint" type="SyncPointType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <xsd:element name="TranscriptSelection" type="
                ↪ TranscriptSelectionType" minOccurs="0" maxOccurs="unbounded"
                ↪ />
            <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="richTextPath" type="xsd:string"/>
        <xsd:attribute name="plainTextPath" type="xsd:string"/>
        <xsd:attribute name="creatingUser" type="GUIDType"/>
        <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
        <xsd:attribute name="modifyingUser" type="GUIDType"/>
        <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
        <!-- Either PlainTextContent or plainTextPath MUST be filled, not both
            ↪ -->
    </xsd:complexType>
    <xsd:complexType name="TranscriptSelectionType">
        <xsd:sequence>
            <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="Coding" type="CodingType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
            <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
        <xsd:attribute name="name" type="xsd:string"/>
        <xsd:attribute name="fromSyncPoint" type="GUIDType"/>
        <xsd:attribute name="toSyncPoint" type="GUIDType"/>
        <xsd:attribute name="creatingUser" type="GUIDType"/>
        <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
        <xsd:attribute name="modifyingUser" type="GUIDType"/>
        <xsd:attribute name="modifiedDateTime" type="xsd:dateTime"/>
    </xsd:complexType>
    <xsd:complexType name="SyncPointType">
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
        <xsd:attribute name="timeStamp" type="xsd:integer"/>
```

```
    <xsd:attribute name="position" type="xsd:integer"/>
</xsd:complexType>
<xsd:complexType name="CodingType">
    <xsd:sequence>
        <xsd:element name="CodeRef" type="CodeRefType"/>
        <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="creatingUser" type="GUIDType"/>
    <xsd:attribute name="creationDateTime" type="xsd:dateTime"/>
</xsd:complexType>
<xsd:complexType name="GraphsType">
    <xsd:sequence>
        <xsd:element name="Graph" type="GraphType" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="GraphType">
    <xsd:sequence>
        <xsd:element name="Vertex" type="VertexType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
        <xsd:element name="Edge" type="EdgeType" minOccurs="0" maxOccurs="
            ↪ unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="VertexType">
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="representedGUID" type="GUIDType"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="firstX" type="xsd:integer" use="required"/>
    <xsd:attribute name="firstY" type="xsd:integer" use="required"/>
    <xsd:attribute name="secondX" type="xsd:integer"/>
    <xsd:attribute name="secondY" type="xsd:integer"/>
    <xsd:attribute name="shape" type="ShapeType"/>
    <xsd:attribute name="color" type="RGBType"/>
</xsd:complexType>
<xsd:complexType name="EdgeType">
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="representedGUID" type="GUIDType"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="sourceVertex" type="GUIDType" use="required"/>
    <xsd:attribute name="targetVertex" type="GUIDType" use="required"/>
    <xsd:attribute name="color" type="RGBType"/>
    <xsd:attribute name="direction" type="directionType"/>
    <xsd:attribute name="lineStyle" type="LineStyleType"/>
</xsd:complexType>
<xsd:complexType name="NotesType">
    <xsd:sequence>
```

```
        <xsd:element name="Note" type="TextSourceType" maxOccurs="unbounded
            ↪ "/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LinksType">
    <xsd:sequence>
        <xsd:element name="Link" type="LinkType" maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="LinkType">
    <xsd:sequence>
        <xsd:element name="NoteRef" type="NoteRefType" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="guid" type="GUIDType" use="required"/>
    <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="direction" type="directionType"/>
    <xsd:attribute name="color" type="RGBType"/>
    <xsd:attribute name="originGUID" type="GUIDType"/>
    <xsd:attribute name="targetGUID" type="GUIDType"/>
</xsd:complexType>
<xsd:complexType name="NoteRefType">
    <xsd:attribute name="targetGUID" type="GUIDType" use="required"/>
</xsd:complexType>
<xsd:complexType name="CodeRefType">
    <xsd:attribute name="targetGUID" type="GUIDType" use="required"/>
</xsd:complexType>
<xsd:complexType name="SourceRefType">
    <xsd:attribute name="targetGUID" type="GUIDType" use="required"/>
</xsd:complexType>
<xsd:complexType name="SelectionRefType">
    <xsd:attribute name="targetGUID" type="GUIDType" use="required"/>
</xsd:complexType>
<xsd:complexType name="VariableRefType">
    <xsd:attribute name="targetGUID" type="GUIDType" use="required"/>
</xsd:complexType>
<xsd:simpleType name="GUIDType">
    <xsd:restriction base="xsd:token">
        <xsd:pattern value="([0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F
            ↪ ]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12})|(\{[0-9a-fA-F]{8}-[0-9a
            ↪ -fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}\})"/
            ↪ >
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="RGBType">
    <xsd:restriction base="xsd:token">
        <xsd:pattern value="#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="directionType">
    <xsd:restriction base="xsd:token">
```

```
            <xsd:enumeration value="Associative"/>
            <xsd:enumeration value="OneWay"/>
            <xsd:enumeration value="Bidirectional"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="typeOfVariableType">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="Text"/>
            <xsd:enumeration value="Boolean"/>
            <xsd:enumeration value="Integer"/>
            <xsd:enumeration value="Float"/>
            <xsd:enumeration value="Date"/>
            <xsd:enumeration value="DateTime"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="ShapeType">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="Person"/>
            <xsd:enumeration value="Oval"/>
            <xsd:enumeration value="Rectangle"/>
            <xsd:enumeration value="RoundedRectangle"/>
            <xsd:enumeration value="Star"/>
            <xsd:enumeration value="LeftTriangle"/>
            <xsd:enumeration value="RightTriangle"/>
            <xsd:enumeration value="UpTriangle"/>
            <xsd:enumeration value="DownTriangle"/>
            <xsd:enumeration value="Note"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="LineStyleType">
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="dotted"/>
            <xsd:enumeration value="dashed"/>
            <xsd:enumeration value="solid"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

Listing 1: Project XML-Schema (van Blommestein, 2019)

B XML-Schema für den Austausch von Codebüchern im REFI-QDA-Standard

```
<xsd:schema xmlns="urn:QDA-XML:codebook:1.0" xmlns:xsd="http://www.w3.org
    ↪ /2001/XMLSchema" targetNamespace="urn:QDA-XML:codebook:1.0"
    ↪ elementFormDefault="qualified" attributeFormDefault="unqualified"
    ↪ version="1.0">
    <!-- ===== Element Declarations ===== -->
    <xsd:element name="CodeBook" type="CodeBookType">
        <xsd:annotation>
            <xsd:documentation>This element MUST be conveyed as the root
                ↪ element in any instance document based on this Schema
                ↪ expression</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <!-- ===== Type Definitions ===== -->
    <xsd:complexType name="CodeBookType">
        <xsd:sequence>
            <xsd:element name="Codes" type="CodesType"/>
            <xsd:element name="Sets" type="SetsType" minOccurs="0"/>
        </xsd:sequence>
        <xsd:attribute name="origin" type="xsd:string"/>
    </xsd:complexType>
    <xsd:complexType name="CodesType">
        <xsd:sequence>
            <xsd:element name="Code" type="CodeType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="SetsType">
        <xsd:sequence>
            <xsd:element name="Set" type="SetType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="CodeType">
        <xsd:sequence>
            <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="Code" type="CodeType" minOccurs="0" maxOccurs="
                ↪ unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
        <xsd:attribute name="isCodable" type="xsd:boolean" use="required"/>
        <xsd:attribute name="color" type="RGBType"/>
    </xsd:complexType>
    <xsd:complexType name="SetType">
        <xsd:sequence>
            <xsd:element name="Description" type="xsd:string" minOccurs="0"/>
            <xsd:element name="MemberCode" type="MemberCodeType" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
        </xsd:sequence>
```

```
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
        <xsd:attribute name="name" type="xsd:string" use="required"/>
    </xsd:complexType>
    <xsd:complexType name="MemberCodeType">
        <xsd:attribute name="guid" type="GUIDType" use="required"/>
    </xsd:complexType>
    <xsd:simpleType name="GUIDType">
        <xsd:restriction base="xsd:token">
            <xsd:pattern value="([0-9a-fA-F]{8}-[0-9a-fA-F]{4}-[0-9a-fA-F
                ↪ ]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12})|(\{[0-9a-fA-F]{8}-[0-9a
                ↪ -fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{4}-[0-9a-fA-F]{12}\})"/
                ↪ >
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="RGBType">
        <xsd:restriction base="xsd:token">
            <xsd:pattern value="#([A-Fa-f0-9]{6}|[A-Fa-f0-9]{3})"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

Listing 2: Codebook XML-Schema (van Blommestein, 2019)

Literaturverzeichnis

van Blommestein, F. (2019). *Exchange of processed data between qualitative data analysis software packages* [PDF file, no direct link anymore, access through download]. https://www.qdasoftware.org/project-implementation-files