Universität Zürich[UZH]

INLINE EDITING IN QFQ

EXPOSEE

Inline-Editing ermöglicht es, den Inhalt einer Webseite direkt in der Ansicht zu bearbeiten, ohne eine separate Bearbeitungsansicht aufrufen zu müssen. Dies ermöglicht eine nahtlose und effiziente Aktualisierung von Texten oder Daten im Kontext der Seite.

Pascal Rössler

Universität Zürich

Institut für Mathematik

Inhaltsverzeichnis

1 Arbeitsdokumentation (Teil 1)

1.1 Beschreibung

QFQ ist eine an der Universität Zürich entwickelte Extension (PHP, JavaScript) für das CMS Typo3.

QFQ unterstützt den Webentwickler in diversen Bereichen; Von einfacher Datenbank-Anbindung über einen Formular-Editor bis hin zu einer eigenen Syntax, die zur Erstellung von dynamischen Seiteninhalten dient. Die Universität Zürich betreibt mehrere Webtools, welche auf QFQ basieren.

Das Erfassen, Anzeigen, Bearbeiten und Löschen (CRUD) von Daten steht dabei oft im Zentrum. Zurzeit können einzelne Datensätze in zuvor erstellten QFQ-Formularen erstellt und bearbeitet werden. Übersichtsseiten, die mehrere Datensätze in Tabellenform anzeigen, können mithilfe der QFQ-Syntax erstellt werden.

Im Rahmen der IPA soll eine Inline-Editing Funktionalität in QFQ implementiert werden, sodass man Zellen von Datensätzen direkt auf der Tabellenansicht bearbeiten kann, ohne zuerst ein dediziertes Formular öffnen zu müssen.

Zur Umsetzung dieser Aufgabe werden PHP, HTML, CSS, JavaScript und SQL eingesetzt.

1.2 Ausgangslage

QFQ ist eine an der Universität Zürich entwickelte Extension (PHP, JavaScript) für das CMS Typo3. Die Universität Zürich betreibt mehrere Webtools, welche auf QFQ basieren.

Im Zentrum von QFQ stehen zwei Aspekte:

1. Ein Formular-Editor, der es erlaubt, Eingabeformulare mit wenig Aufwand zu erstellen. Die damit erstellten QFQ Formulare werden mit Tabellen aus der Datenbank in Verbindung gebracht, sodass Benutzereingaben direkt am richtigen Ort abgespeichert werden.
2. Eine eigene Syntax, mit der Seiteninhalte dynamisch aufgebaut werden können (QFQ-Reports), mit direktem Zugriff auf die SQL-Datenbank.

QFQ-Reports werden oft verwendet, um eine Übersicht über mehrere Datensätze in Tabellenform anzuzeigen. Um einen Datensatz zu bearbeiten, muss dieser aber in einem QFQ-Formular geöffnet werden. Das bedeutet für den Anwender zusätzliche Klicks zum Öffnen, Speichern und Schliessen des Formulars. Oft wäre es sinnvoller, wenn die Daten direkt in der Tabelle bearbeitet werden könnten (Inline-Editing).

1.3 Aufgabenstellung

Im Rahmen der IPA soll das QFQ-Framework so erweitert werden, dass Inline-Editing von Daten möglich wird.

1.3.1 Ziele

QFQ soll eine neue Special Column* "_edit" zur Verfügung stellen, die ein Inline-Edit-Element erzeugt.

Das Inline-Edit-Element ist beim Laden der Seite ein gewöhnliches Label mit einem onclick-Event.

Klickt der Benutzer darauf, dann wird eine Anfrage an den Server gesendet, der dann eine Textbox ausliefert.

Das zuvor angezeigte Label wird ausgeblendet und an derselben Stelle wird die Textbox eingefügt.

Die Textbox erhält automatisch den Fokus.

Verliert diese wieder den Fokus, dann wird eine weitere Anfrage an den Server gemacht, um die Änderung zu speichern.

Es wird keine Anfrage gemacht, wenn der Wert nicht verändert wurde.

In jedem Fall wird aber die Textbox ausgeblendet und das Label wird mit dem neuen Wert aktualisiert, bevor dieses wieder eingeblendet wird.

Der Server antwortet auf die Speichern-Anfrage mit einem "ok" oder einem "error".

Im Falle eines Errors muss auch das angezeigte Label auf den ursprünglichen Wert zurückgesetzt werden.

Wird dasselbe Label erneut angeklickt, dann soll die bestehende Textbox verwendet/eingeblendet werden, anstatt eine neue Textbox beim Server anzufragen.

Wird ein Datensatz über ein Inline-Edit-Element verändert, dann soll diese Änderung geloggt werden. (I5)

- Geloggt werden soll in die Tabelle FormSubmitLog, die bereits bei QFQ-Formulareingaben verwendet wird.
- Die Tabellendefinition darf erweitert werden, falls dies zur Nachvollziehbarkeit der Eingabe beiträgt.

Falls die Anwendung der Special Column Name zu einer Exception führt, dann soll dem Benutzer (= QFQ-Webentwickler) im Browser eine möglichst aussagekräftige Fehlermeldung angezeigt werden.

Das Exception Handling soll folgendes abdecken:

- Weder Schema der Variante 1 noch Schema der Variante 2 eingehalten
- Angegebenes Formular oder Formularelement existiert nicht (Variante 1)
- Angegebene Tabelle oder Spalte existiert nicht (Variante 2)
- Angegebener Datensatz (r) existiert nicht

** **Special Column**: Ein reserviertes QFQ-Keyword. Wird dieses Keyword als Spaltenalias innerhalb eines von QFQ ausgelösten SQL-SELECT-Statements gewählt, dann wird spezifische Funktionalität auslöst.*

Die Special Column* "_edit" soll auf zwei Arten (I1) verwendet werden können:

1. Referenzierend auf ein QFQ-FormElement**, nach dem Schema:
   SELECT 'form=[...]&fe=[...]&r=[...]' AS _edit
   - Nach "form=" soll der Name des QFQ-Formulars angegeben werden, auf welchem sich das FormElement befindet, dessen Name nach "fe=" spezifiziert wird.
   - "r" steht für die id des Datensatzes, der bearbeitet werden soll. Auch r=0 soll möglich sein, um einen neuen Datensatz zu erstellen.
   - Die Tabelle und Spalte auf der Datenbank, der Typ des Input-Elements (muss nur für type=text funktionieren), sowie zusätzliche Parameter sind dabei in der Definition des Forms/Formelements enthalten.
   - Für die Bewertung ist nicht relevant, ob die zusätzlichen Parameter im "Inline-Edit" Modus funktionieren.

2. Direkte Angabe der Tabelle und Spalte, nach dem Schema:
   SELECT 'table=tableName&column=colName&type=text&r=myId' AS _edit
   - Das Inline-Edit-Element ist mit dieser Herangehensweise nicht an ein bestehendes QFQ-FormElement gebunden.
   - Damit eignet es sich nur für einfache Fälle, die keine zusätzlichen Parameter erfordern.

Beide Varianten sollen auf bestehendem Code aufbauen, um das HTML des Input-Elements aufzubauen.

Im Rahmen der IPA muss nur der Fall "type=text" bearbeitet werden. Das Input-Element wird also in jedem Fall eine einfache Textbox sein.

Der Code soll möglichst leicht erweiterbar sein, dass in Zukunft Inline-Editing für weitere Input-Typen (Dropdowns, Datepicker, File-Upload...) implementiert werden kann.

** **Special Column**: Ein reserviertes QFQ-Keyword. Wird dieses Keyword als Spaltenalias innerhalb eines von QFQ ausgelösten SQL-SELECT-Statements gewählt, dann wird spezifische Funktionalität auslöst.*

*** **QFQ-Formelement**: Ein Element innerhalb eines QFQ-Eingabeformulars (Textfeld, Dropdown, etc.). Kann durch den QFQ-Formulareditor erstellt und in der Datenbank gespeichert werden.*

1.3.2 Erwartete Resultate

PHP-Code: Implementierung der Special Column "_edit", möglichst Vollständige Abdeckung der genannten Ziele (I2, I3, I6)

JavaScript-Code: Script für die Clientseite, dass die Kommunikation mit dem Server und das Anzeigen/Verstecken der Labels/Input-Elemente handhabt (I2, I6)

PHP-Unit-Tests: Möglichst gute Abdeckung aller Fälle, die bei der Anwendung der Special Column "_edit" auftreten können (I4, I6)

Technische Dokumentation: Muss die Klassenstruktur und die Kommunikation zwischen Server und Client beschreiben (I1)

Integrations-Tests: Erstellen eines Testplans. Manuelle Durchführung der Testfälle und dokumentieren der Ergebnisse in einem Testprotokoll.

Benutzerhandbuch: Die Anwendung der neuen Special Column soll in der offiziellen QFQ-Dokumentation beschrieben werden (https://docs.qfq.io/en/master/Report.html#special-column-names). Zielgruppe ist hierbei ein Web-Entwickler, der/die mit QFQ arbeitet und das Framework bereits kennt. (I7)

1.4 Mittel und Methoden

- PHP (PHPStorm)
- JavaScript
- HTML
- CSS
- MySQL
- Typo3
- Gitlab
- Linux

1.5 Vorkenntnisse

Der Lernende erstellt QFQ-Webtools seit August 2022.

QFQ-Code beinhaltet typischerweise viele SQL-Abschnitte, daher sind auch gute SQL-Vorkenntnisse vorhanden.

Seit Dezember 2022 aktive Mitarbeit am QFQ-Projekt mit PHP.

Mit JavaScript hat der Lernende noch nicht allzu viel gemacht, ist aber auch kein komplettes Neuland. Bei Fragen stehen der VF, das Team und das Internet zur Verfügung.

1.6 Vorarbeiten

Die Anforderungen und das Konzept wurden im Team erstellt (so wie in detaillierter Aufgabenstellung beschrieben).

1.7 Arbeiten in den letzten 6 Monaten

(Weiter-)Entwicklung der Seite "MedTool" mit QFQ. Sowohl QFQ-Formulare als auch QFQ-Reports wurden erstellt.

Kleine Arbeiten am QFQ-Framework selbst (PHP, JavaScript)

1.8 Projektaufbauorganisation

1.9 Termine

Was	Datum/Zeit
IPA-Dauer	26.05.23-09.06.23 – 80h
Erster Besuchstag Experte	26.05.23 - 8:00
Zweiter Besuchstag Experte	06.06.23 - 8:00
Abgabe IPA Bericht	09.06.23
Präsentation, Demo und Fachgespräch	16.06.23

1.10 Projektorganisation

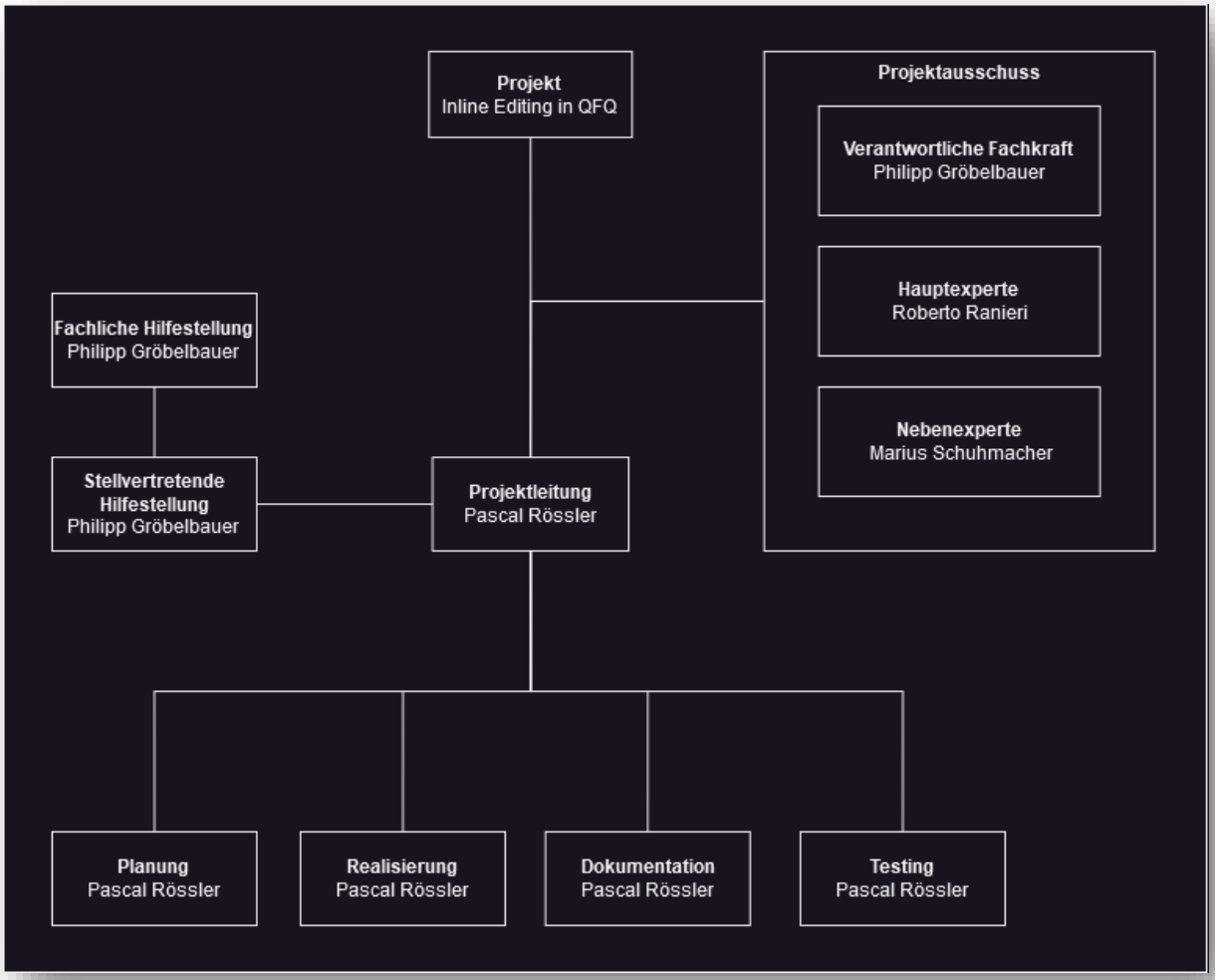


Abbildung 1: Organigramm

1.11 Zeitplan GANTT

Projektphasen	Total				FR 26.05.23	DI 30.05.23	MI 31.05.23	DO 01.06.23	FR 02.06.23	MO 05.06.23	DI 06.06.23	MI 07.06.23	DO 08.06.23	FR 09.06.23
	Soll		Ist											
	Stunden	Prozent	Stunden	Prozent										
Projektzeit	80	100%	80	100%										
Informieren	3	3.75%	3	3.75%										
Detaillierte Aufgabenstellung studieren	1	1.25%	1	1.25%										
Kriterienkatalog/QV-Leitfaden studieren	1	1.25%	1	1.25%										
Recherche zum Thema Inline-Editing	1	1.25%	1	1.25%										
Planen	7	8.75%	8	10%										
Zeitplan erstellen	2	2.5%	1	1.25%										
Use Case Diagramm erstellen	1	1.25%	1	1.25%										
Klassen Diagramm erstellen	2	2.5%	3	3.75%										
Sequenz Diagramm erstellen	2	2.5%	3	3.75%										
Entscheiden	1	1.25%	1	1.25%										
Design Pattern definieren	1	1.25%	1	1.25%										
Realisieren	36	45%	29	36.25%										
InlineEdit.php Klasse implementieren	20	25%	14	17.5%										
API/Schnitstelle erstellen	2	2.5%	4	5%										
JavaScript implementieren	6	7.5%	3	3.75%										
Fehlerbehandlung implementieren	2	2.5%	2	2.5%										
Logging implementieren	5	6.25%	5	6.25%										
Benutzeranleitung schreiben (QFQ-Doc)	1	1.25%	1	1.25%										
Kontrollieren	7	8.8%	4	5%										
Testkonzept erarbeiten	1	1.25%	1	1.25%										
Unit-Test erstellen	4	5%	3	3.75%										
Integrations-Test erstellen	2	2.5%	1	1.25%										
Auswerten	2	2.5%	2	2.5%										
Zeitplanung SOLL/IST abgleichen	1	1.25%	1	1.25%										
Reflexion und Schlusswort schreiben	1	1.25%	1	1.25%										
Dokumentation & Organisation	24	30%	33	41.25%										
Gespräch mit Experten	2	2.5%	2	2.5%										
Dokumentation Struktur erstellen	1	1.25%	1	1.25%										
Dokumentation/Tagesjournal schreiben	19	23.75%	30	37.5%										
Reserve	2	2.5%	0	0%										

Abbildung 2: Zeitplan GANTT

1.12 Arbeitsjournal

1.12.1 Tag 1, Freitag, 26. April 2023

1.12.1.1 Geplante Tätigkeiten

Projektphase	Aufgabe	Soll-Zeit (h)
Organisation	Gespräch mit Experten	1
Informieren	Detaillierte Aufgabenstellung studieren	1
Informieren	Kriterienkatalog/QV-Leitfaden studieren	1
Informieren	Recherche zum Thema Inline Editing	1
Planen	Zeitplan erstellen	2
Dokumentation	Dokumentation Struktur erstellen	1
Dokumentation	Tagesjournal schreiben	1

1.12.1.2 Erledigte Tätigkeiten

Projektphase	Aufgabe	Ist-Zeit (h)
Organisation	Gespräch mit Experten	1
Informieren	Detaillierte Aufgabenstellung studieren	1
Informieren	Kriterienkatalog/QV-Leitfaden studieren	1
Informieren	Recherche zu Thema Inline Editing	1
Planen	Zeitplan erstellen	1
Dokumentation	Dokumentation Struktur erstellen	1.5
Planen	Use Case Diagramm erstellen	1
Dokumentation	Tagesjournal schreiben	0.5

1.12.1.3 Bemerkungen (Erfolge, Probleme, Entscheidungen)

Erreichte Ziele	Das erste Expertengespräch fand, wie geplant statt und ich konnte wertvolle Informationen und Erkenntnisse für mein Projekt sammeln. Ich habe die detaillierte Aufgabenstellung sowie den Kriterienkatalog und den QV-Leitfaden gründlich studiert und dadurch ein besseres Verständnis für die Anforderungen an mein Projekt gewonnen. In meiner Recherche zum Thema Inline Editing habe ich mir hauptsächlich angeschaut, wie "phpMyAdmin" Inline Editing umgesetzt hat. Obwohl ich ursprünglich zwei Stunden für die Erstellung des Zeitplans eingeplant hatte, habe ich es in einer Stunde geschafft. Trotzdem bin ich mit dem Ergebnis zufrieden und habe einen strukturierten Zeitplan erstellt. Die Dokumentationsstruktur wurde wie geplant erstellt und bietet eine solide Grundlage für meine weiteren Aufgaben. Zusätzlich zu meinen geplanten Aktivitäten habe ich auch ein Use Case Diagramm erstellt.
Probleme	Heute gab es keine besonderen Probleme oder Schwierigkeiten. Die Aktivitäten verliefen größtenteils reibungslos und ich konnte meine Ziele wie geplant erreichen.
Reflexion	Heute war ein produktiver Tag. Das Gespräch mit den Experten war wertvoll, da ich Informationen erhalten habe, die mir bei der weiteren Planung und Umsetzung meines Projekts helfen werden. Ich habe erkannt, wie wichtig es ist, genügend Zeit für das Studium der relevanten Materialien aufzuwenden. Die Erstellung des Zeitplans war ein wichtiger Schritt, um meine Aufgaben zu strukturieren. Insgesamt bin ich zuversichtlich, dass ich auf dem richtigen Weg bin, um meine Projektziele erfolgreich zu erreichen.

1.12.1.4 Hilfestellung

Wer?	Frage	Antwort

1.12.2 Tag 2, Dienstag, 30. April 2023

1.12.2.1 Geplante Tätigkeiten

Projektphase	Aufgabe	Soll-Zeit (h)
Planen	Use Case Diagramm erstellen	1
Planen	Klassen Diagramm erstellen	2
Planen	Sequenz Diagramm erstellen	2
Entscheiden	Design Pattern entscheiden	1
Dokumentation	Dokumentation schreiben	1.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.2.2 Erledigte Tätigkeiten

Projektphase	Aufgabe	Ist-Zeit (h)
Planen	Klassen Diagramm erstellen	1
Planen	Sequenz Diagramm erstellen	1
Entscheiden	Design Pattern entscheiden	1
Realisieren	InlineEdit.php implementieren	3
Dokumentation	Dokumentation schreiben	1.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.2.3 Bemerkungen (Erfolge, Probleme, Entscheidungen)

Erreichte Ziele	Als erstes habe ich angefangen das Klassen Diagramm zu erstellen um die verschiedenen Klassen und ihre Beziehungen zueinander dargestellt. Danach widmete ich mich dem Sequenzdiagramm, das mir eine detaillierte Darstellung der Interaktionen zwischen den verschiedenen Objekten des Systems ermöglichen soll. Zudem habe ich mir Gedanken dazu gemacht wie ich meinen Code strukturieren will und mich für den Factory Design Pattern entschieden. Die Implementation der "InlineEdit.php" Klasse habe ich damit begonnen den "Constructor" und die Properties der Klasse zu definieren. Jeder Schritt hat mir geholfen ein besseres Verständnis des Systems zu entwickeln und meine Arbeit strukturiert und dokumentiert voranzutreiben.
Probleme	Es gab keine größeren Probleme während der Arbeit an den einzelnen Aufgaben. Jedoch hatte ich die Erkenntnis, dass ich nach der Implementierung höchstwahrscheinlich die Diagramme anpassen werden muss. Daher habe ich beschlossen, heute weniger Zeit für die Diagrammerstellung einzuplanen und stattdessen schon mit der Implementierung zu starten.
Reflexion	Insgesamt konnte ich die meisten meiner geplanten Tätigkeiten erfolgreich abschließen. Die Entscheidung, weniger Zeit für die Diagrammerstellung einzuplanen, erwies sich als sinnvoll, da dies Zeit für die Implementierung freimachte. Ich habe erkannt, dass es wichtig ist, flexibel zu sein und gegebenenfalls die Zeitplanung anzupassen, um effizienter arbeiten zu können. Die Erfahrung, die Diagramme vor der Implementierung zu erstellen, hat mir geholfen, einen besseren Überblick über das Projekt zu bekommen und mögliche Probleme im Voraus zu identifizieren. Die Dokumentation und das Tagesjournal wurden wie geplant weitergeführt. Insgesamt bin ich zufrieden mit meinen erreichten Zielen und habe wertvolle Erkenntnisse für zukünftige Projekte gewonnen.

1.12.2.4 Hilfestellung

Wer?	Frage	Antwort
P.Gröbelbauer	Muss berücksichtigt werden, wenn auf einer Seite zweimal der gleiche Datensatz angezeigt wird, dass der zweite Record erst beim Seiten Laden den neuen Wert bekommt?	Muss nicht berücksichtigt werden
P.Gröbelbauer	In Ordnung das Input-Element Textarea ist?	Ja

1.12.3 Tag 3, Mittwoch, 31. April 2023

1.12.3.1 Geplante Tätigkeiten

Projektphase	Aufgabe	Soll-Zeit (h)
Realisieren	InlineEdit.php Klasse implementieren	6
Dokumentation	Dokumentation schreiben	1.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.3.2 Erledigte Tätigkeiten

Projektphase	Aufgabe	Ist-Zeit (h)
Realisieren	InlineEdit.php Klasse implementieren	6
Dokumentation	Dokumentation schreiben	1.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.3.3 Bemerkungen (Erfolge, Probleme, Entscheidungen)

Erreichte Ziele	Heute habe ich mich mit der Einbindung und Fertigstellung der Einstiegsfunktion "process()" in meinem Projekt beschäftigt. Diese Funktion ermöglicht es mir, basierend auf den übergebenen Parametern, ein Prepared Statement für eine SQL-Abfrage zu erstellen, um von einem Formnamen auf einen Tabellennamen zu gelangen. Des Weiteren habe ich die Funktion "render()" implementiert, um den HTML-Code für das Label und den Container zu erzeugen. Zusätzlich konnte ich dem Container-Element das HTML-Attribut "data-sip" mit sip-codierten Werten übergeben, die später zur Erzeugung des Eingabefelds benötigt werden. Zudem habe ich mir stichwortartig Punkte für die Dokumentation notiert zu denen ich an einem späteren Zeitpunkt detaillierter beschreiben werde.
Probleme	Was mir heute ein wenig Probleme gemacht hat war die Implementierung der Funktionalitäten zur sip-Codierung und -Decodierung. Um die Werte sip-codiert abzulegen und später wieder darauf zugreifen zu können, verwendete ich zunächst die Funktion "queryStringToSip()" für die Codierung und die Funktion "getVarsFromSip()" für die Decodierung. Allerdings stellte ich fest, dass diese beiden Funktionen nur funktionieren, wenn sie auf derselben Instanz codiert und decodiert werden.
Reflexion	Insgesamt bin ich zufrieden mit den Fortschritten, die ich heute erzielt habe. Trotz der Herausforderungen bei der sip-Codierung und -Decodierung konnte ich die Einstiegsfunktion "process()" erfolgreich integrieren und die Funktion "render()" implementieren.

1.12.3.4 Hilfestellung

Wer?	Frage	Antwort
P.Gröbelbauer	Welche Naming-Convention sollen die HTML Attribute "id" und "name" befolgen?	Klein Buchstaben, durch Bindestrich trennen, eindeutige Bezeichnungen.

1.12.4 Tag 4, Donnerstag, 1. Juni 2023

1.12.4.1 Geplante Tätigkeiten

Projektphase	Aufgabe	Soll-Zeit (h)
Realisieren	InlineEdit.php Klasse implementieren	7
Dokumentation	Dokumentation schreiben	0.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.4.2 Erledigte Tätigkeiten

Projektphase	Aufgabe	Ist-Zeit (h)
Realisieren	InlineEdit.php Klasse implementieren	5
Realisieren	API/Schnittstelle erstellen	2
Dokumentation	Dokumentation schreiben	0.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.4.3 Bemerkungen (Erfolge, Probleme, Entscheidungen)

Erreichte Ziele	Heute habe ich erfolgreich die Funktion "createFe()" implementiert. Diese Funktion ermöglicht es, ein sogenanntes "fake" FormElement auf Basis der im "data-sip" hinterlegten Werte zu erzeugen. Das erzeugte FormElement wird zuerst als HTML gerendert und dann über die API-Schnittstelle als Eingabefeld zur Verfügung gestellt. Mit diesem Schritt konnte ich also heute schon die InlineEdit.php Klasse fertigstellen und bereits mit der API-Schnittstelle beginnen.
Probleme	Die Erstellung der API-Schnittstelle war anfangs etwas schwierig, da ich an einigen Stellen im Code nicht über die notwendigen Variablen verfügte. Daher musste ich Zeit damit verbringen, mir bereits existierende Schnittstellen anzuschauen und wie diese mein Problem lösten.
Reflexion	Ich habe festgestellt, dass ich an bestimmten Stellen in der Codebase, wie zum Beispiel den Klassen "FormElementInput" und "FormElementRenderBS3", weniger Schwierigkeiten hatte. Dies liegt daran, dass ich diese bereits im Vorfeld oft gebraucht habe. Im Gegensatz dazu habe ich bei der Implementierung der API-Schnittstelle nicht so schnell Fortschritte erzielt, da ich bisher wenig Erfahrung mit diesem Teil unserer Codebase hatte. Alles in allem bin ich mit dem heutigen Arbeitstag sehr zufrieden, da ich bereits heute die Implementierung der "InlineEdit.php" Klasse abschliessen konnte.

1.12.4.4 Hilfestellung

Wer?	Frage	Antwort
P.Gröbelbauer	Kann die Fehlerbehandlung der SQL-Statements (DbExceptions), mit der bereits existierenden Lösung ohne weitere Anpassungen realisiert werden?	Ja

1.12.5 Tag 5, Freitag, 2. Juni 2023

1.12.5.1 Geplante Tätigkeiten

Projektphase	Aufgabe	Soll-Zeit (h)
Realisieren	InlineEdit.php Klasse implementieren	7
Dokumentation	Dokumentation schreiben	0.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.5.2 Erledigte Tätigkeiten

Projektphase	Aufgabe	Ist-Zeit (h)
Realisieren	API/Schnittstelle erstellen	2
Realisieren	JavaScript implementieren	3
Realisieren	Fehlerbehandlung implementieren	2
Dokumenation	Dokumentation schreiben	0.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.5.3 Bemerkungen (Erfolge, Probleme, Entscheidungen)

Erreichte Ziele	Da ich bereits gestern mit der Implementierung der API-Schnittstelle begonnen habe, konnte ich diesen Teil bereits heute abschließen. Außerdem habe ich den JavaScript-Teil fertiggestellt, der für die Generierung des Eingabefeldes zuständig ist. Dazu konnte ich die Funktionalität implementieren, dass nur wenn der Benutzer auf ein Label geklickt hat, ein Eingabefeld "on the fly" erzeugt werden soll. Dazu noch die Funktionalität um anschließend das Eingabefeld und das Label ein- und auszublenden. Danach hatte ich noch Zeit aussagekräftige Fehlermeldungen, für die Angabe des "special column name _edit", zu implementieren.
Probleme	Mit dem Teil der API-Schnittstelle hatte ich wieder Probleme, die ich aber lösen konnte, indem ich den ganzen Ablauf mit dem Debugger genau analysierte und mit der Hilfestellung von P.Gröbelbauer und C.Rose. Das Problem war, dass die Klasse "QuickFormQuery.php", die von unseren API-Klassen instanziiert wird, davon ausgeht, dass ein Formular existiert, aber da ich nur ein "fake" Formularelement erstellt habe, musste ich einige Parameter für ein "fake" Formular definieren, um die Funktionen der Klasse nutzen zu können.
Reflexion	Ich bin mit den Fortschritten, die ich heute gemacht habe, sehr zufrieden. Ich freue mich, dass ich den JavaScript-Teil und die API-Schnittstelle erfolgreich abschließen konnte. Überrascht war ich, dass implementieren des JavaScripts einfacher war als erwartet. Die Schwierigkeiten, auf die ich gestoßen bin, haben mich gezwungen, tiefer in den Code einzutauchen. Diese Erfahrungen sind wertvoll für meine persönliche Entwicklung als Entwickler. Mich motiviert es, dass ich dem Zeitplan voraus bin und, dass meiste schneller als geplant abschliessen konnte.

1.12.5.4 Hilfestellung

Wer?	Frage	Antwort
P.Gröbelbauer, C.Rose	Wie wurde bis jetzt der Sip Store befüllt und danach über den API-Aufruf wiedergeholt? (schwer zu verstehender Legacy-Code).	Die Klasse bietet die Funktion "renderLink()", mit dem Parameter "r:8" können Werte im Sip Store abgelegt werden und der Sip Schlüssel wird direkt erzeugt. Beim Ajax Call soll die Variable "dataSip" in der URL mitgeliefert werden und nicht im "data: " Feld. Dann soll über ein "QuickFormQuery" Objekt die "doForm()" Funktion aufgerufen werden um Zugriff auf die im Sip Store abgelegten Werte zu erhalten.

1.12.6 Tag 6, Montag, 5. Juni 2023

1.12.6.1 Geplante Tätigkeiten

Projektphase	Aufgabe	Soll-Zeit (h)
Realisieren	API/Schnittstelle implementieren	2
Realisieren	JavaScript Implementieren	5
Dokumentation	Dokumentation schreiben	0.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.6.2 Erledigte Tätigkeiten

Projektphase	Aufgabe	Ist-Zeit (h)
Realisieren	Logging implementieren	5
Realisieren	Benutzeranleitung schreiben (QFQ-Doc)	1
Dokumentation	Dokumentation schreiben	1.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.6.3 Bemerkungen (Erfolge, Probleme, Entscheidungen)

Erreichte Ziele	Heute stand vor allem das Logging im Fokus. Da in unserer Codebase bereits eine Funktion existiert, die beim Speichern eines Formulars Einträge in die "FormSubmitLog" Tabelle macht, konnte ich diese für meinen Fall benutzten. Zudem musste ich eine kleine Anpassung in der Tabellenstruktur vornehmen. Ich habe eine neue Spalte erstellt mit der Nachvollziehbar gemacht werden soll, ob es sich bei einem Record der via Inline Editing bearbeitet wurde, handelt oder nicht. Dann habe ich noch die Benutzeranleitung geschrieben in der die Nutzung, des "special column name _edit" erklärt wird. Zudem habe ich noch ein paar Beispiele eingefügt, sodass der QFQ-Entwickler diese einfach kopieren und anpassen können.
Probleme	Am meisten Probleme hat mir heute gemacht, Zugriff auf benötigte Variablen wie die "pageId" zu erhalten. Da diese von dem CMS Typo3 verwaltet werden, musste ich eine Weile suchen, bis ich eine Lösung gefunden habe.
Reflexion	Obwohl ich das Schreiben der Dokumentation bis jetzt etwas vernachlässigt habe, konnte ich dennoch gute Fortschritte bei der Umsetzung des eigentlichen Projekts erzielen. Darum habe ich mich heute dafür entschieden mehr Zeit als geplant in die Dokumentation zu investieren. Zudem habe ich die Implementation unterschätzt, um alle nötigen variabel vorzubereiten musste ich mehr Zeit als zuerst erwartet in die Recherche und das Debugging zur Lösung des Problems investieren.

1.12.6.4 Hilfestellung

Wer?	Frage	Antwort

1.12.7 Tag 7, Dienstag, 6. Juni 2023

1.12.7.1 Geplante Tätigkeiten

Projektphase	Aufgabe	Soll-Zeit (h)
Organisation	Gespräch mit Experten	1
Realisieren	JavaScript implementieren	1
Realisieren	Fehlerbehandlung implementieren	2
Realisieren	Logging implementieren	3
Dokumentation	Dokumentation schreiben	0.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.7.2 Erledigte Tätigkeiten

Projektphase	Aufgabe	Ist-Zeit (h)
Organisation	Gespräch mit Experten	1
Kontrollieren	Testkonzept erarbeiten	1
Kontrollieren	Unit-Test erstellen	3
Kontrollieren	Integrations-Test erstellen	1
Planen	Klassen Diagramm erstellen	1
Dokumentation	Dokumentation schreiben	0.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.7.3 Bemerkungen (Erfolge, Probleme, Entscheidungen)

Erreichte Ziele	Ich habe mich ausführlich mit der Erstellung von Unit-Tests beschäftigt. Dabei habe ich gelernt, wie man Mocks von Klassen erstellt, um die Daten vorzubereiten. Dies hat mir geholfen, die Funktionalität meiner einzelnen Komponenten zu überprüfen und potenzielle Fehler frühzeitig zu erkennen. Insgesamt habe ich einen detaillierten und umfassenden Unit-Tests für mein Projekt entwickelt. Ich hatte bereits mit der Erstellung eines Klassen-Diagramms begonnen und konnte es heute erfolgreich abschließen.
Probleme	Heute ist alles reibungslos und so wie ich mir das vorgestellt habe nach vorwärts gegangen.
Reflexion	Die heutigen Tätigkeiten waren sehr produktiv und haben mir wertvolle Erkenntnisse gebracht. Besonders das Erstellen der Unit-Tests und das Arbeiten mit Mocks waren eine wertvolle Lernerfahrung. Ich habe gelernt, wie wichtig es ist, die einzelnen Komponenten meiner Software sorgfältig zu testen, um mögliche Fehler frühzeitig zu erkennen. Zudem habe ich die Bedeutung einer gut strukturierten Dokumentation erkannt, um meinen Fortschritt nachvollziehen zu können und anderen Entwicklern einen Einblick in mein Projekt zu ermöglichen. Insgesamt bin ich sehr zufrieden mit dem Fortschritt meines Projektes, da ich dem geplanten Zeitplan voraus bin.

1.12.7.4 Hilfestellung

Wer?	Frage	Antwort
P.Gröbelbauer	Klassendiagram Feedback, Use Case Feedback	Es sollte klar sichtbar sein welche Klassen von mir programmiert wurden und welche bereits vorher existierten. Im Use Case Diagramm sollte nur mein Feature sichtbar sein unnötige Zusatzinformationen soll ich löschen.

1.12.8 Tag 8, Mittwoch, 7. Juni 2023

1.12.8.1 Geplante Tätigkeiten

Projektphase	Aufgabe	Soll-Zeit (h)
Realisieren	Logging implementieren	2
Realisieren	Benutzeranleitung schreiben (QFQ-Doc)	1
Kontrollieren	Testkonzept erarbeiten	1
Kontrollieren	Unit-Test erstellen	2
Dokumentation	Dokumentation schreiben	1.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.8.2 Erledigte Tätigkeiten

Projektphase	Aufgabe	Ist-Zeit (h)
Planen	Sequenzdiagramm erstellen	2
Auswerten	Zeitplanung SOLL/IST abgleichen	1
Auswerten	Reflexion und Schlusswort schreiben	1
Dokumentation	Dokumentation schreiben	3.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.8.3 Bemerkungen (Erfolge, Probleme, Entscheidungen)

Erreichte Ziele	Heute konnte ich den grössten Teil meiner Arbeit abschliessen, nun kann ich mich voll auf die Dokumentation konzentrieren. Ich habe meine Zeitplanung mit dem tatsächlichen Fortschritt meines Projekts abgeglichen. Dabei habe ich festgestellt, dass ich meine Zeitressourcen effektiv genutzt habe und im Allgemeinen im Rahmen meiner Planung geblieben bin. Ich habe Zeit für eine umfassende Reflexion über meine Tätigkeiten des Tages genommen. Dabei habe ich meine Erfolge, Herausforderungen und Erkenntnisse reflektiert und ein abschließendes Schlusswort verfasst. Diese Reflexion hilft mir, meine Stärken und Schwächen zu erkennen und zukünftige Verbesserungen in meiner Arbeitsweise zu identifizieren.
Probleme	Aufgrund von Änderungen in meiner Code-Implementierung musste ich das Sequenzdiagramm komplett neu erstellen. Dies hat zusätzliche Zeit und Aufwand erfordert. In Zukunft werde ich versuchen, frühzeitig genug abzuschätzen, ob meine ursprüngliche Planung mit meinen Implementierungsänderungen übereinstimmt, um solche Überarbeitungen zu vermeiden.
Reflexion	Die heutigen Tätigkeiten waren anspruchsvoll, aber lohnend. Ich habe wichtige Meilensteine erreicht, wie das Abschließen des Sequenzdiagramms und die Überprüfung meiner Zeitplanung. Die Reflexion hat mir geholfen, meine Fortschritte und Herausforderungen zu erkennen und daraus zu lernen. Insbesondere habe ich die Bedeutung einer flexiblen Planung und regelmäßigen Abgleiche zwischen SOLL und IST erkannt.

1.12.8.4 Hilfestellung

Wer?	Frage	Antwort
P.Gröbelbauer	Sequenzdiagramm Feedback	Die Lifelines waren zum Teil nicht richtig beendet.

1.12.9 Tag 9, Donnerstag, 8. Juni 2023

1.12.9.1 Geplante Tätigkeiten

Projektphase	Aufgabe	Soll-Zeit (h)
Kontrollieren	Unit-Test erstellen	2
Kontrollieren	Integrations-Test erstellen	2
Auswerten	Zeitplanung SOLL/IST abgleichen	1
Auswerten	Reflexion und Schlusswort schreiben	1
Dokumentation	Dokumentation schreiben	1.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.9.2 Erledigte Tätigkeiten

Projektphase	Aufgabe	Ist-Zeit (h)
Dokumentation	Dokumentation schreiben	7.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.9.3 Bemerkungen (Erfolge, Probleme, Entscheidungen)

Erreichte Ziele	Ich konnte den grössten Teil des Kapitels Realisieren für meine Dokumentation abschliessen. Ich bin von Anfang bis Ende des Programmablaufs durchgegangen und habe alle relevanten Stellen mit Code-Snippet Screenshots und Erklärungen erläutert.
Probleme	Das Einzige, was man als Problem ansehen könnt, wäre das ich beim Schreiben der Dokumentation nicht so gut wie geplant vorwärtsgekommen bin und jetzt alles nachholen muss.
Reflexion	Geplante wäre gewesen, dass ich einen grösseren Teil der Dokumentation vorzu abschliessen kann, jedoch war ich sehr darauf fokussiert den Realisieren Teil meiner Arbeit abzuschliessen können. Daher konnte ich mich Heute voll auf das Schreiben der Dokumentation fokussieren.

1.12.9.4 Hilfestellung

Wer?	Frage	Antwort

1.12.10 Tag 10, Freitag, 9. Juni 2023

1.12.10.1 Geplante Tätigkeiten

Projektphase	Aufgabe	Soll-Zeit (h)
Dokumentation	Dokumentation schreiben	5.5
Dokumentation	Tagesjournal schrieben	0.5
Organisation	Reserve	2

1.12.10.2 Erledigte Tätigkeiten

Projektphase	Aufgabe	Ist-Zeit (h)
Dokumentation	Dokumentation schreiben	7.5
Dokumentation	Tagesjournal schreiben	0.5

1.12.10.3 Bemerkungen (Erfolge, Probleme, Entscheidungen)

Erreichte Ziele	Am letzten Tag angekommen, konnte ich mich vollkommen auf das Fertigstellen der Dokumentation konzentrieren. Ich bin alle Bewertungskriterien durchgegangen und habe somit kontrolliert ob alle Kriterien, durch meine Projektarbeit, erfüllt sind. Danach habe ich die Arbeit gründlich auf Rechtschreibe- und Formulierfehler kontrolliert und diese ausgebessert. Zudem habe ich Probeuploads, auf "pkorg", gemacht, um sicher zu stellen, dass alles richtig funktioniert. Danach kann ich stolz behaupten, dass ich zufrieden mit meiner Projektarbeit fertig wurde.
Probleme	Heute traten keine Probleme auf, ich hatte sogar noch ein wenig Zeit zu viel. Die ich in das allgemeine schöner machen meiner Dokumentation gesteckt habe.
Reflexion	Dass ich mit dem Realisieren der Arbeit so schnell fertig geworden bin, hat mir enorm die Anspannung gelöst, da ich keine Angst hatten musste, dass ich nicht fertig werde. Zudem fand ich das Schreiben der Dokumentation ziemlich anstrengend. Ich bin mir nicht sicher, ob es besser gewesen wäre, wenn ich vorzu mehr Zeit in die Dokumentation gesteckt hätte.

1.12.10.4 Hilfestellung

Wer?	Frage	Antwort

2 Projektdokumentation (Teil 2)

2.1 Summary

Während meines Praktikums habe ich, mit QFQ, Tools entwickelt und dabei festgestellt, dass ich mir eine Inline-Editing-Funktion wünschen würde. Deshalb wusste ich sofort, als mir dieses Feature als IPA-Thema vorgeschlagen wurde, dass ich es umsetzen will. Um das neue Feature zu implementieren, habe ich zunächst über Inline-Editing recherchiert und untersucht, wie andere dieses Feature implementiert haben. Außerdem studierte ich die detaillierte Aufgabenstellung.

Anschließend habe ich die gesamte Implementierung gründlich geplant, indem ich die Klassenstruktur in einem Klassendiagramm dargestellt und den Programmablauf in Sequenzdiagrammen festgehalten habe. Für die Erstellung des Input-Elements habe ich mich für das Factory Design Pattern entschieden, um das bereits vorhandene "FormElementInput", das von der abstrakten Klasse "AbstractFormElement" erbt, zu instanziieren. Dadurch kann das Inline-Editing-Feature relativ einfach auf andere Eingabetypen erweitert werden.

Der HTML-Code des Features besteht aus einem Containerelement und einem Label, das den zu editierenden Wert umschließt. Alle Parameter, die für die Aktualisierung der Datensätze erforderlich sind, sind im "Sip Store" abgelegt und der "Sip Key" wird im HTML-Attribut "data-Sip" gespeichert. Wenn der Benutzer nun auf diesen Container klickt, wird über das JavaScript-Modul "InlineEdit.js" ein "Ajax-Post-Request" an den Server gesendet, der das Eingabefeld an der richtigen Stelle liefert. Für das JavaScript habe ich "jQuery" verwendet und zusätzliche Funktionen implementiert, um das Verhalten der "Textarea" für den Benutzer so benutzerfreundlich wie möglich zu machen. Wenn der Benutzer den Wert der "Textarea" ändert und das Eingabefeld den Fokus verliert, wird erneut ein "Ajax-Request" an den Server gesendet, der den Datensatz aktualisiert, einen Log-Eintrag erstellt und als Antwort auf diesen "Ajax-Request" den neuen Wert in das Label einfügt. Das Ergebnis meiner Arbeit ist das neue Inline Editing Feature für die Typo3 Extension QFQ. QFQ-Entwickler können damit den Benutzern ihres Tools/Website ermöglichen, Datensätze zu editieren, ohne eine separate Bearbeitungsschicht verwenden zu müssen. Es gibt zwei Varianten, "Reference" und "Direct", die dem QFQ-Entwickler zwei verschiedene Möglichkeiten bieten, das neue Feature zu implementieren. Entweder kann durch direkte Angabe von "Tabelle", "Spalte", "Typ" und "ID" oder durch referenzierende Angabe von "Form", "FormElement" und "ID" das Feature implementiert werden.

Zusätzlich wird jede Änderung eines Datensatzes durch das Inline Editing Feature in der "FormSubmitLog" Tabelle protokolliert. Um die Verwendung so verständlich wie möglich zu machen, habe ich ein Benutzerhandbuch für die offizielle QFQ-Dokumentation geschrieben. Außerdem habe ich eine aussagekräftige Fehlerbehandlung implementiert, um QFQ-Entwicklern die korrekte Anwendung des Features zu erleichtern. Zudem habe ich umfangreiche "phpUnit"-Tests geschrieben, um sicherzustellen, dass alles richtig funktioniert.

2.2 Projektorganisation

2.2.1 Projektmethodik

Ich habe mich für **IPERKA** als Projektorganisationsmethode entschieden, weil ich damit sicherstellen kann, dass ich alle wichtigen Schritte unternehme, um mein Projekt erfolgreich abzuschließen. Die Methode gibt mir klare Anweisungen und Strukturen, um den Ablauf meines Projekts zu verbessern und Risiken zu minimieren.

Zunächst ist es wichtig, dass ich mich gründlich informiere. Das bedeutet, dass ich alle relevanten **Informationen** sammle und verstehe, was das Projekt beinhaltet und was meine Ziele sind.

Dann kommt die **Planungsphase**. Hier erstelle ich einen detaillierten Plan, in dem ich festlege, welche Schritte und Aufgaben notwendig sind, um das Projekt zu realisieren.

Wenn der Plan steht, geht es darum, **Entscheidungen** zu treffen. Das bedeutet, dass ich klare Entscheidungen darüber treffe, wie ich gewisse Tätigkeiten umsetzte und welche Prioritäten gesetzt werden müssen.

Wenn die Entscheidungen getroffen sind, geht es an die **Realisation**. Ich arbeite aktiv an den im Plan definierten Aufgaben und Schritten und setze alles daran, die Projektziele zu erreichen.

Während des gesamten Projektverlaufs ist eine kontinuierliche **Kontrolle** sehr wichtig. Ich überprüfe regelmäßig, ob das Projekt planmäßig verläuft und leite bei Abweichungen oder Problemen sofort Maßnahmen ein. Durch diese Kontrolle kann ich frühzeitig reagieren und das Projekt wieder auf den richtigen Weg bringen.

Schließlich kommt die **Auswertung**. Hier bewerte ich das abgeschlossene Projekt, um zu sehen, ob die Ziele erreicht wurden und welche Erfahrungen ich daraus ziehen kann. Ich analysiere die Ergebnisse und mache mir Notizen für zukünftige Projekte.

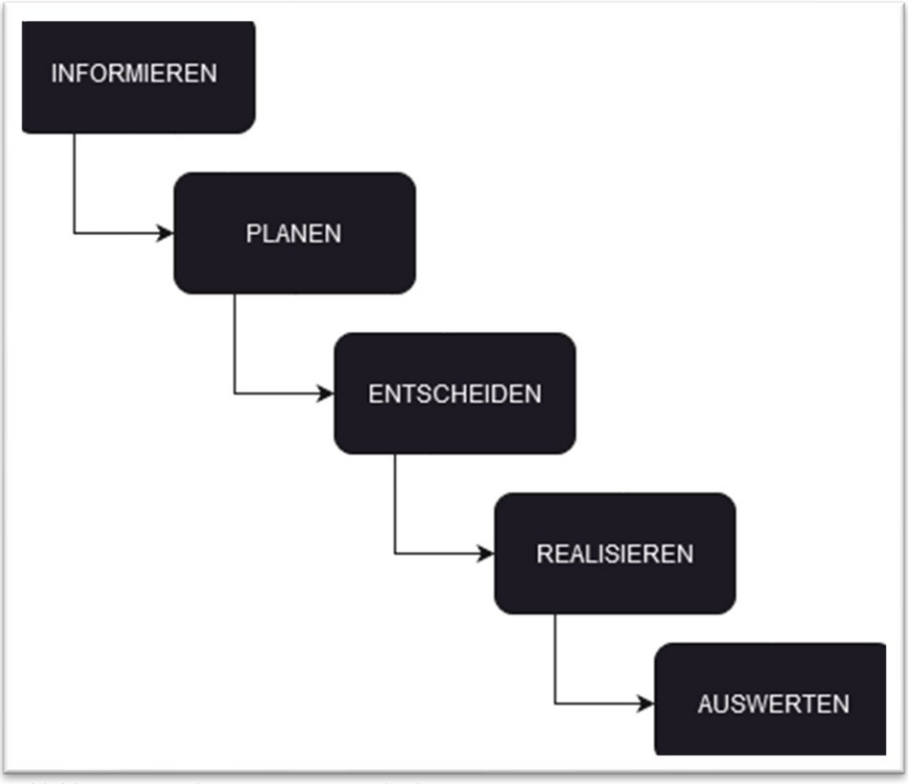


Abbildung 3: Projektorganisationsmethode IPERKA

2.3 Umgebung und Tools

2.3.1 Projektumfeld

2.3.1.1 Komponentendiagramm

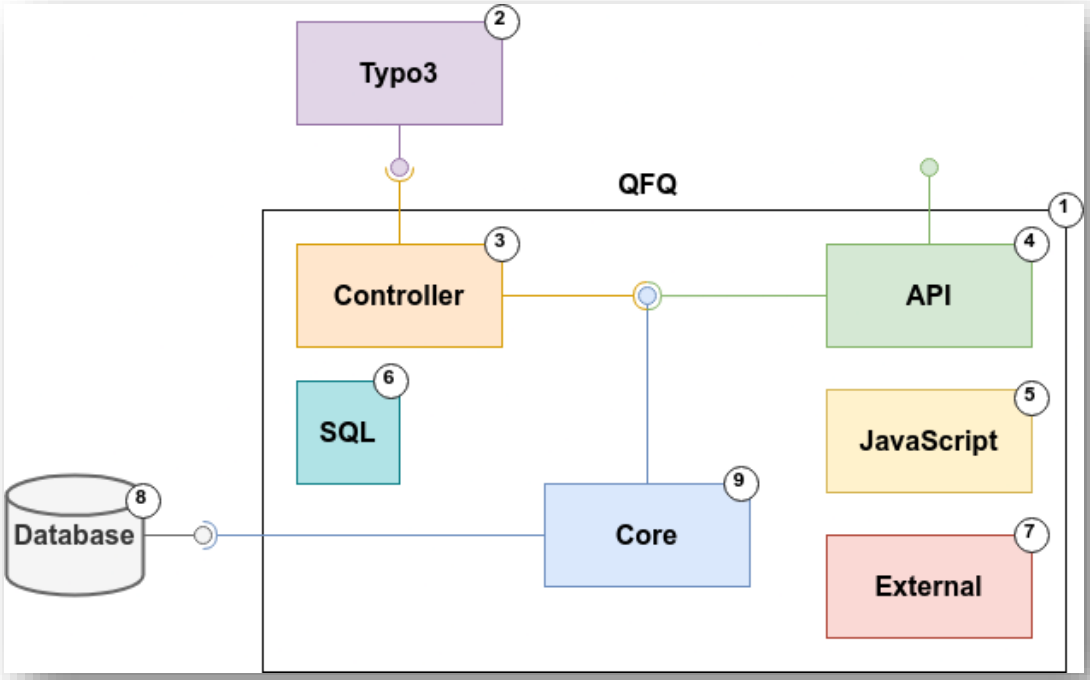


Abbildung 4: Komponentendiagramm

1. Das System **QFQ** besteht wie oben dargestellt aus mehreren Komponenten, die unteranderem dafür zuständig sind mit anderen Systemen zu interagieren, intern Daten zu verarbeiten oder diese wieder auszugeben.

2. Ein solches anderes System wäre das Content Management System **Typo3**.

3. Der **Controller** ist dafür zuständig QFQ mit Typo3 zu verbinden.

4. Über **API**-Scripts werden Schnittstellen, um mit anderen Systemen zu interagieren definiert.

5. Beim Laden eines QFQ-Reports werden sämtliche **JavaScript** Module zu einem qfq.min.js zusammengeführt, minimiert und ausgeliefert.

6. Der Komponent **SQL** enthält diverse SQL-Scripts die zum Beispiel, bei der Erstinstallation erforderliche Tabellen erstellen.

7. Im **External** Komponent befindet sich "AutoCron.php", "sendMail(Perl-Skript) ", "hashPassword.php".

8. Ein weiteres System ist die **Database**, sie ist an dem Core angebunden, sodass Daten abgelegt, abgefragt und manipuliert werden können.

9. Der **Core** ist der zentrale Teil, der die grundlegenden Funktionen und Mechanismen bereitstellt. Er ist wie das Herz des Systems und sorgt dafür, dass alles ordnungsgemäß funktioniert.

2.3.2 Versionierung und Backups

Die Sicherung meiner Projektarbeit spielt eine entscheidende Rolle, um Datenverluste zu verhindern. Unvorhergesehene Ereignisse wie Hardwarefehler oder Softwareprobleme können dazu führen, dass wertvolle Projektinformationen unwiederbringlich verloren gehen. Durch die Implementierung eines effektiven Backup-Systems kann ich das Risiko minimieren und stelle sicher, dass meine Arbeit jederzeit wiederherstellbar ist.

GitLab-Commits für die Versionierung:

GitLab bietet eine leistungsstarke Versionierungsfunktion, die es mir ermöglicht, meine Projektarbeit in Form von Commits auf dem Branch "F16305_Inline_Editing_Text" zu verfolgen. Durch regelmäßige Commits erfass ich die Änderungen, die vorgenommen werden, und kann jederzeit auf frühere Versionen zurückgreifen.

USB-Stick als Datenträger für die tägliche Dokumentationssicherung:

Um eine zusätzliche Sicherungsebene zu gewährleisten, verwende ich einen USB-Stick als Datenträger, um täglich die Word-Dokumentation abzulegen. Dabei lege ich die Dateien in entsprechenden Tagesordnern ab, um eine geordnete Struktur und leichtere Wiederherstellbarkeit zu gewährleisten

Regelmäßige Backups der ThinLinc Session:

Zusätzlich zu den individuellen Sicherungsmaßnahmen werden automatisch Snapshots der "ThinLinc Session" gemacht die ebenfalls als Backups dienen.

2.4 Informieren

2.4.1 Ist-Situation

Bis jetzt wurde in allen, von uns mit QFQ realisierten, Tools eine separate Bearbeitungsschicht benötigt, um Änderungen an Datensätzen vorzunehmen. An dem häufig vorkommenden Beispiel einer Tabelle zur Verwaltung von Personen lässt sich gut zeigen was genau damit gemeint ist.

1. Im TYPO3-Backend wird ein QFQ-Content-Element erstellt, das für jede Zeile in der Tabelle einen "Bearbeiten"-Button generiert.

```
form={{form:SE}}
10{
  sql = SELECT CONCAT('p:{{pageSlug:T}}?form=Person&r=',id,'|b|s|E') AS _link
              , firstName
              , lastName
              , address
        FROM Person LIMIT 5

  head = <table class="qfq-table-50">
           <thead>
               <tr>
               <th>{{'p:{{pageSlug:T}}?form=Person&r=0|s|N' AS _link}}</th>
               <th>First Name</th>
               <th>Last Name</th>
               <th>Address</th>
             </tr>
           </thead>
           <tbody>
  rbeg = <tr>
  fbeg = <td>
  fend = </td>
  rend = </tr>
  tail = </tbody></table>
}
```

Abbildung 5: Beispiel QFQ-Content-Element

2. Wenn der Benutzer auf einen solchen Button klickt, wird ein Formular geöffnet, dass den zu bearbeitenden Datensatz enthält.

	First Name	Last Name	Address
	Pascal	Meier	Strasse 2
	Sebastian	Muster	Weg 3
	Ludwig	Keller	Gasse 5
	Sarah	Kuster	Tunnel 7
	Nina	Tester	Strasse 5

Abbildung 6: Beispiel Personen Tabelle

3. In diesem Formular können dann die entsprechenden Daten des Datensatzes bearbeitet werden. Nachdem die Bearbeitung abgeschlossen ist, kann der Benutzer entweder das Formular automatisch schließen lassen oder es manuell schließen.

Person

First Name	Pascal
Last Name	Meier
Address	Strasse 2

Abbildung 7: Beispiel QFQ-Form

2.5 Planen

2.5.1 Use Cases

Die unten aufgeführte Abbildung stellt die, für dieses Projekt relevanten, Use Cases einer Typo3 Page dar.

Der Akteur **QFQ-Entwickler** ist an folgenden Prozessen beteiligt:

- Erstellen eines neuen "QFQ Content Element"
- Daten anzeigen
- Dem Benutzer die Bearbeitung von Daten erlauben
- "QFQ Form" einrichten
- Benutzung des "special colum name: _edit"

Der Akteur **Website-User** ist and folgenden Prozessen beteiligt:

- Daten ansehen
- Daten bearbeiten
- "QFQ Form" benutzen
- Inline Editing benutzen

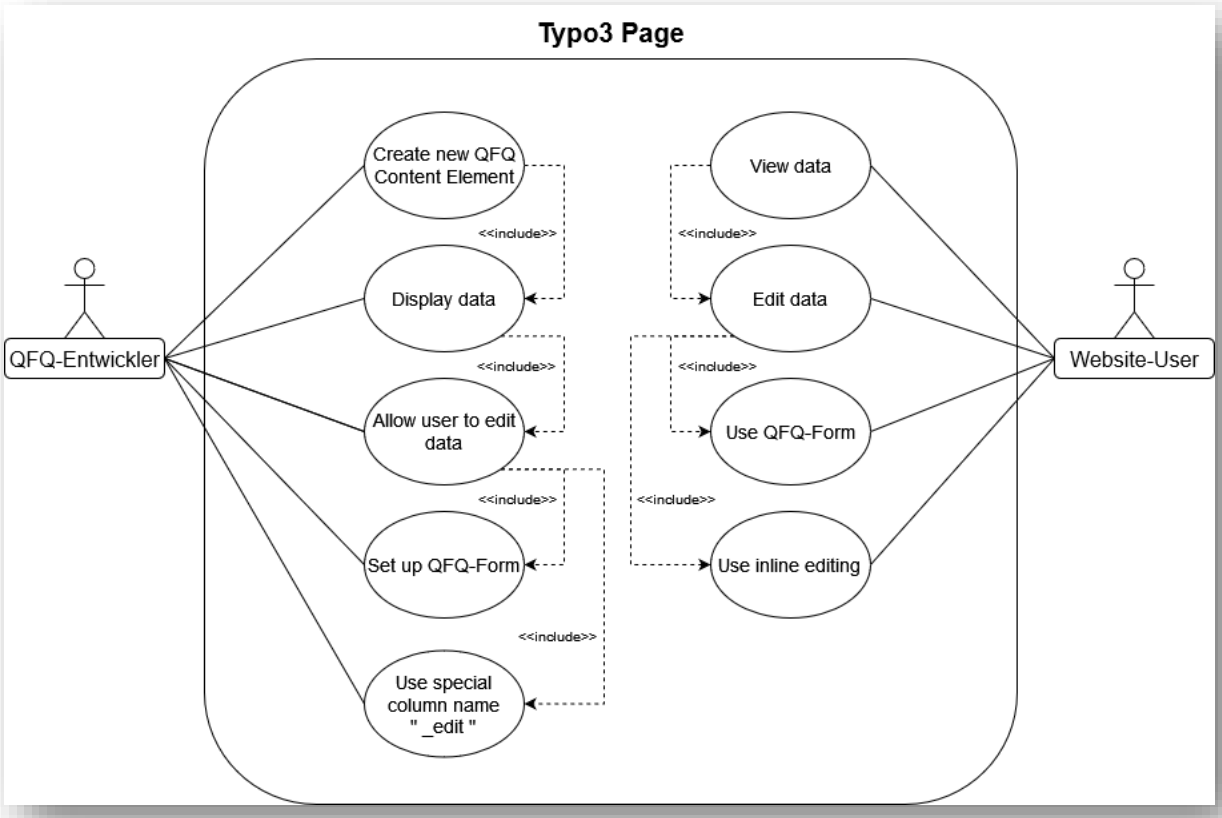


Abbildung 8: Use Case Diagramm

2.5.2 Sequenzdiagramme

2.5.2.1 "Load Page" Sequenz

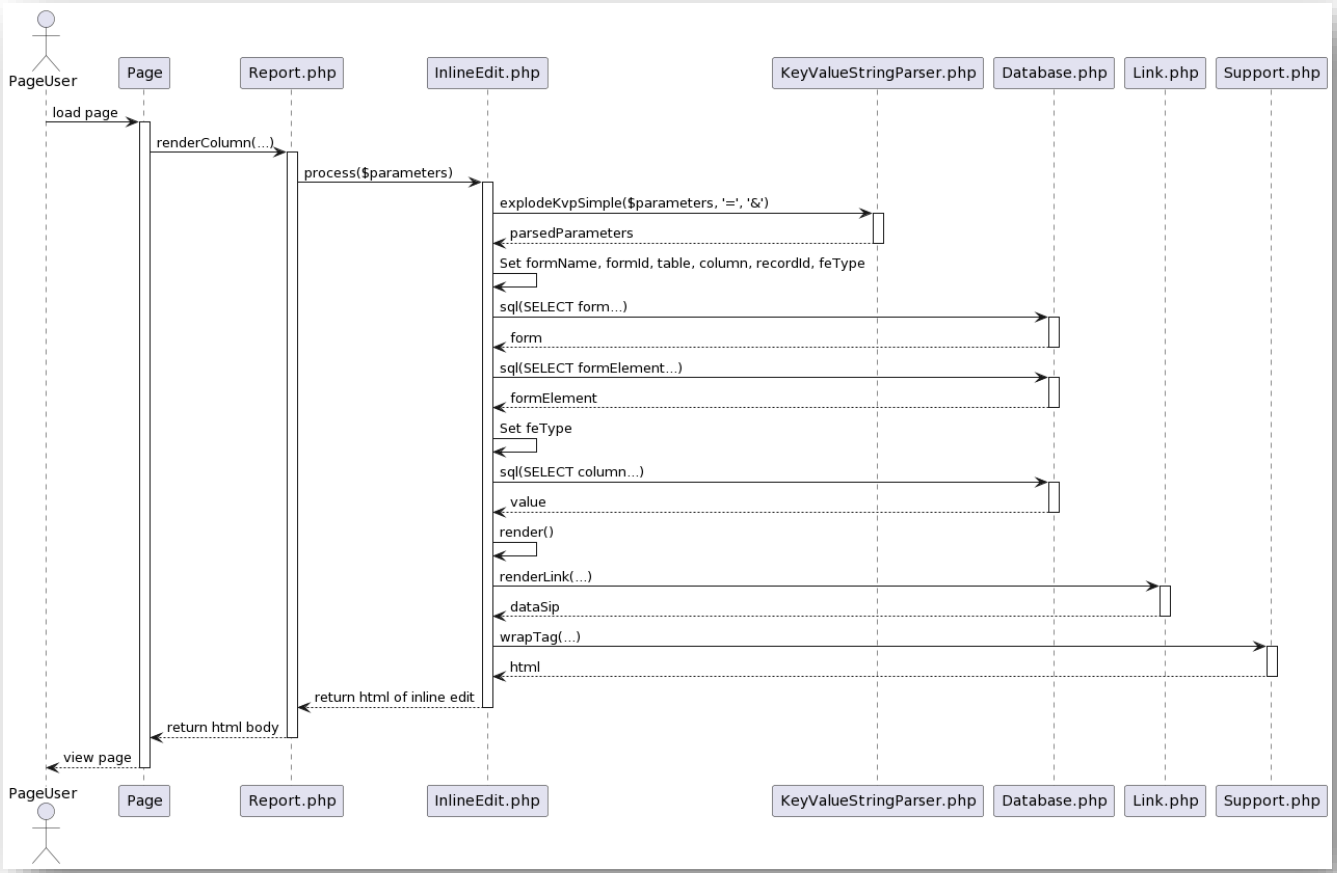


Abbildung 9: Sequenzdiagramm 1

1. Der Benutzer lädt die Webseite.
2. "Report.php" wird aufgerufen, um eine Spalte zu rendern.
3. "InlineEdit.php" verwendet "KeyValueStringParser.php", um die übergebenen Parameter zu parsen.
4. Nach der Umwandlung gibt "KeyValueStringParser.php" die geparsten Parameter an "InlineEdit.php" zurück.
5. Es werden in der "InlineEdit.php" Instanz verschiedene Properties basierend auf den geparsten Parametern gesetzt.
6. "InlineEdit.php" ruft "Database.php" auf, um Daten aus der Datenbank abzurufen, einschließlich des Formulars, der Formularelemente und des aktuellen Werts der Spalte.
7. "InlineEdit.php" erhält die Informationen aus der Datenbank und setzt Properties.
8. "InlineEdit.php" ruft die Funktion "render()" auf, um den HTML-Code zu erzeugen.
9. "InlineEdit.php" ruft "Link.php" auf, um einen Link zu rendern, hierbei werden essenzielle Information im "Sip-Store" abgelegt.
10. "Link.php" gibt den Schlüssel zu den Informationen in Form eines "Sip's" an "InlineEdit.php" zurück.
11. "InlineEdit.php" ruft "Support.php" auf, um HTML-Tags zu erstellen.
12. "Support.php" gibt das HTML an "InlineEdit.php" zurück.
13. "InlineEdit.php" gibt das gerenderte HTML an "Report.php" zurück.
14. "Report.php" gibt den HTML-Body an die Webseite zurück.
15. Die Webseite wird angezeigt, und der Benutzer sieht den Wert der angegebenen Spalte des entsprechenden Datensatzes.

2.5.2.2 "Generate Textbox" Sequenz

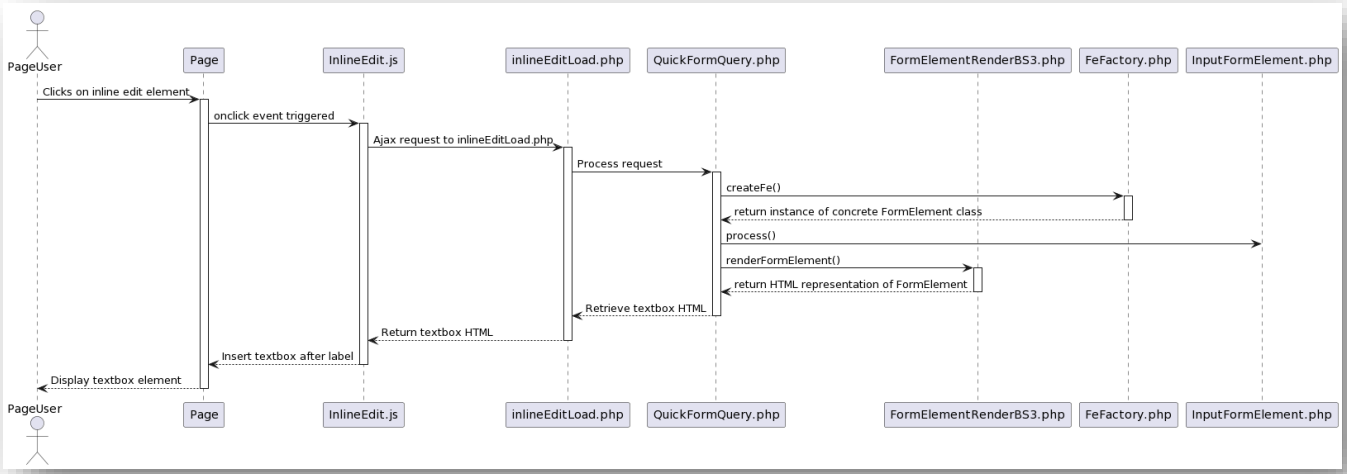


Abbildung 10: Sequenzdiagramm 2

1. Der Benutzer klickt auf ein Inline-Bearbeitungselement auf der Webseite.
2. Die Webseite reagiert auf das Klickereignis und aktiviert das JavaScript-Modul "InlineEdit.js".
3. "InlineEdit.js" sendet eine Ajax-Anfrage an "inlineEditLoad.php", um die Inhalte für die Inline-Bearbeitung zu laden.
4. "inlineEditLoad.php" verarbeitet die Anfrage und ruft "QuickFormQuery.php" auf.
5. "QuickFormQuery.php" verwendet die Factory-Klasse "FeFactory.php", um eine konkrete Klasse abzurufen, die von der abstrakten Klasse "AbstractFormElement" erbt.
6. Die Factory-Klasse "FeFactory.php" erstellt mit der Funktion "createFe" eine Instanz der "InputFormElement.php" Klasse und gibt sie an "QuickFormQuery.php" zurück.
7. "QuickFormQuery.php" ruft "FormElementRenderBS3.php" auf, um das "FormElement" in HTML zu rendern.
8. "QuickFormQuery.php" erhält das gerenderte HTML des Textfelds von "FormElementRenderBS3.php".
9. "QuickFormQuery.php" gibt das gerenderte HTML an "inlineEditLoad.php" zurück.
10. "inlineEditLoad.php" gibt das HTML des Textfelds an "InlineEdit.js" zurück.
11. "InlineEdit.js" fügt das Textfeld nach dem entsprechenden Label in die Webseite ein.
12. Die Webseite zeigt das Textfeld für die Inline-Bearbeitung dem Benutzer an.

2.5.2.3 "Update Record" Sequenz

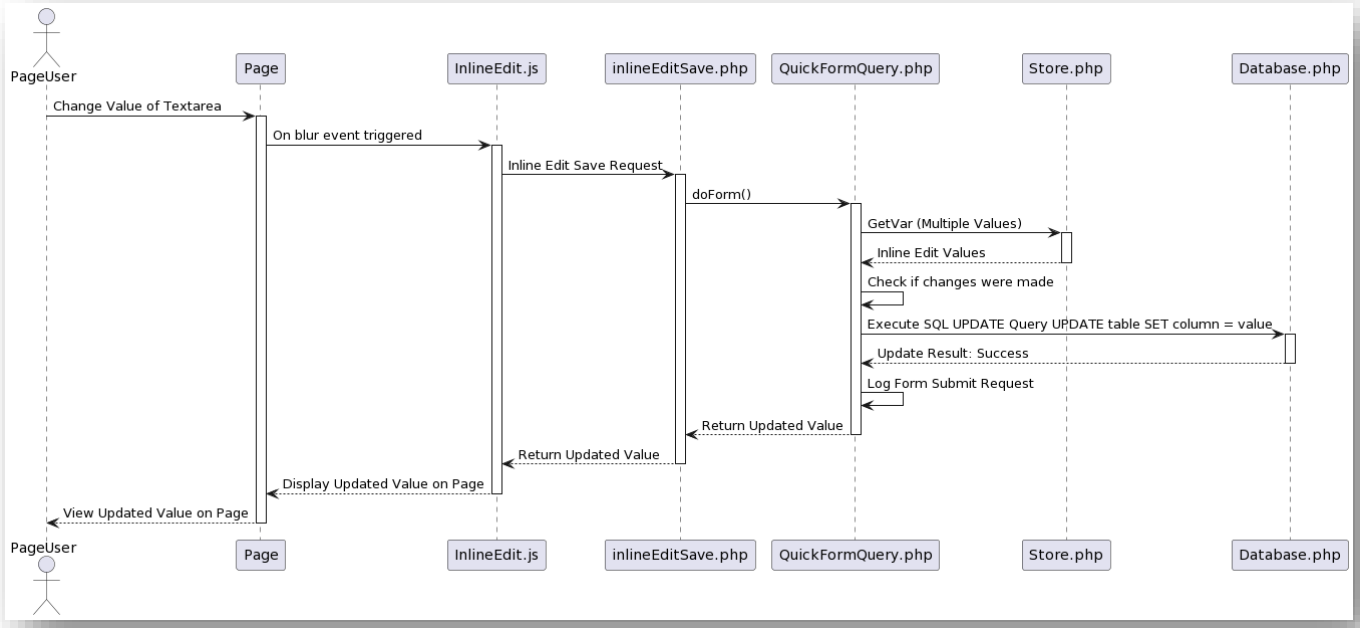


Abbildung 11: Sequenzdiagramm 3

1. Der Benutzer ändert den Wert eines Textfelds auf der Webseite via Inline Editing.
2. Die Webseite erfasst die Änderung und reagiert auf das Auslösen des "blur"-Ereignisses des Textfelds.
3. Das JavaScript-Modul "InlineEdit.js" wird aktiviert und empfängt das "blur"-Ereignis.
4. "InlineEdit.js" sendet eine Anfrage (Inline Edit Save Request) an "inlineEditSave.php", um die Änderung zu speichern.
5. "inlineEditSave.php" aktiviert und ruft "QuickFormQuery.php" auf.
6. "QuickFormQuery.php" ruft "Store.php" auf, um den geänderten Wert zu erhalten.
7. "QuickFormQuery.php" überprüft, ob überhaupt Änderungen vorgenommen wurden.
8. Wenn Änderungen vorgenommen wurden, führt "QuickFormQuery.php" das "SQL UPDATE" Statement durch, um den entsprechenden Datensatz mit dem neuen Wert zu aktualisieren.
9. "QuickFormQuery.php" zeichnet den Erfolg der Aktualisierung auf.
10. Im Falle einer Änderung wird ein Eintrag in die "FormSubmitLog" Tabelle gemacht.
11. Das Ergebnis der Aktualisierung wird von "QuickFormQuery.php" an "inlineEditSave.php" zurückgegeben.
12. "inlineEditSave.php" gibt das Ergebnis an "InlineEdit.js" zurück.
13. "InlineEdit.js" erhält das Ergebnis und aktualisiert die Anzeige auf der Webseite.
14. Die Webseite zeigt mit einem Label den aktualisierten Wert des Textfelds dem Benutzer an.
15. Der Benutzer sieht den neuen Wert auf der Seite.

2.5.3 Klassendiagramm

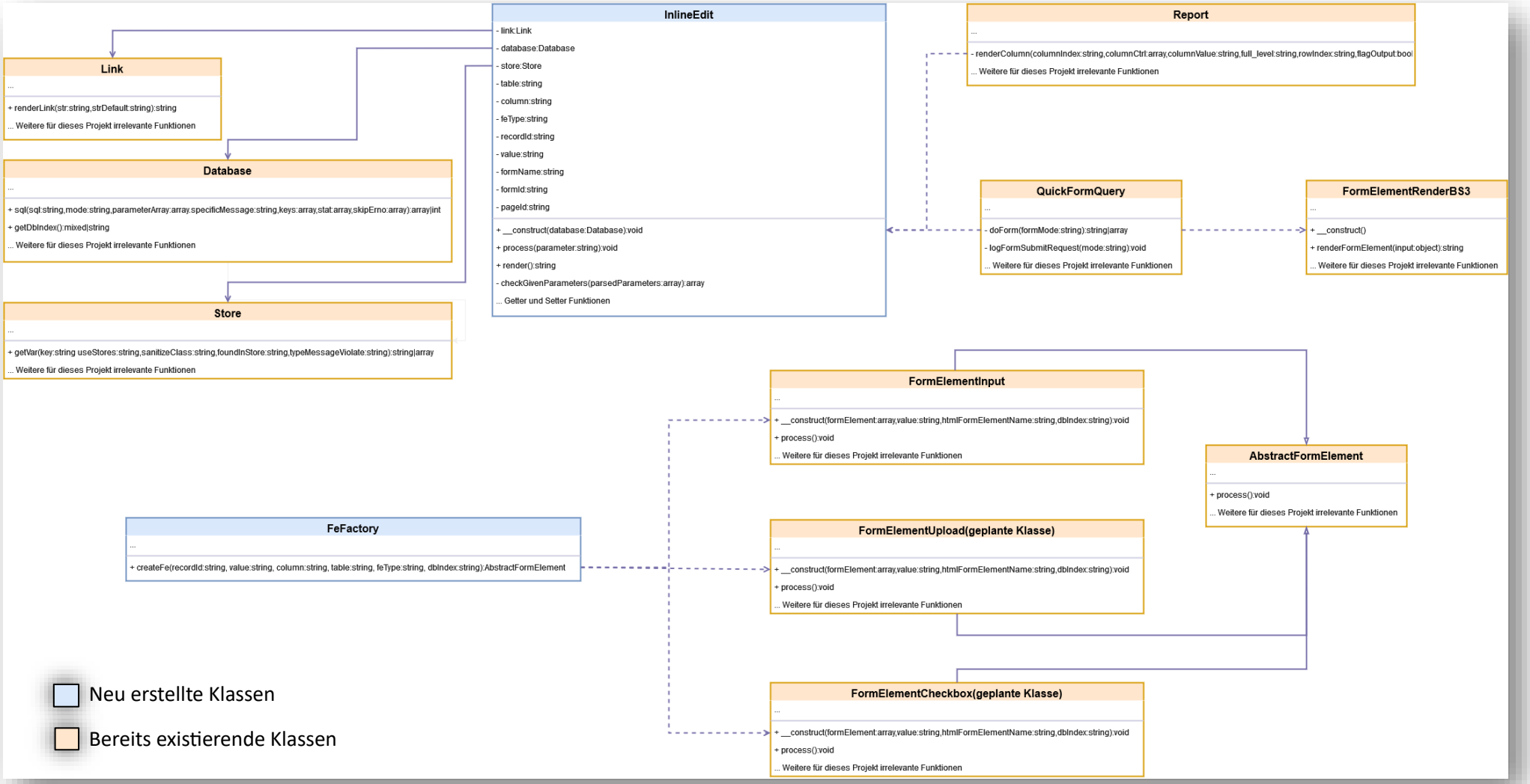


Abbildung 12: Klassendiagramm

2.6 Entscheiden

2.6.1 Design Pattern

2.6.1.1 Factory Pattern

Für die Erstellung von "FormElement" Objekten habe ich mich für das Factory Design Pattern entschieden, da es sich optimal für die Erstellung verschiedener konkreter FormElement-Klassen eignet, die von der abstrakten Klasse "AbstractFormElement" erben. Das Factory Design Pattern ermöglicht es uns, die Erstellung von Objekten zu zentralisieren, wodurch der Code flexibler und erweiterbarer wird.

Indem wir eine Factory-Klasse wie FeFactory verwenden, können wir die Verantwortung der Objekterstellung von den aufrufenden Klassen trennen. Anstatt die konkreten FormElement-Klassen direkt zu instanziieren, rufen wir die Methode createFe der FeFactory auf, um eine Instanz der entsprechenden konkreten Klasse basierend auf dem angegebenen Formularelementtyp zu erhalten.

Dies ermöglicht es uns, neue konkrete FormElement-Klassen hinzuzufügen, ohne den vorhandenen Code ändern zu müssen. Wenn wir beispielsweise in Zukunft neue Formularelementtypen für das Inline Editing Feature anbieten wollen wie zum Beispiel "checkbox" oder "radio", können wir einfach eine neue konkrete Klasse erstellen, die von AbstractFormElement erbt, und die entsprechende Logik in der Factory-Klasse hinzufügen, ohne den Code in anderen Klassen zu modifizieren.

2.6.1.2 Arrange, Act, Assert Unit Test Design Pattern

Für das Testen meines neuen Features habe ich mich für das Unit-Test Design Pattern "Arrange -> Act -> Assert" entschieden. Dieses Muster bietet eine klare Struktur und ermöglicht uns eine systematische Durchführung von Tests für einzelne Komponenten oder Funktionen.

Durch die Verwendung des "Arrange -> Act -> Assert"-Musters kann ich meinen "Testing-Code" gut strukturieren und die Lesbarkeit verbessern. Der Ablauf des Tests wird in drei Schritte unterteilt:

1. **Arrange**: In diesem Schritt werden die erforderlichen Vorbedingungen für den Test vorbereitet. Das umfasst das Einrichten des Testumfelds, das Instanziieren von Objekten, das Definieren von Testdaten und das Festlegen der erwarteten Ergebnisse.

2. **Act**: In diesem Schritt wird die eigentliche Aktion ausgeführt, die getestet werden soll. Das kann das Aufrufen einer Methode, das Auslösen eines Ereignisses oder das Durchführen einer spezifischen Aktion.

3. **Assert**: In diesem Schritt wird das Ergebnis der Aktion überprüft und es wird es mit den erwarteten Ergebnissen verglichen. Hier werden Prüfungen durchgeführen, ob bestimmte Bedingungen erfüllt sind, ob die zurückgegebenen Werte korrekt sind oder ob bestimmte Zustände erreicht wurden.

2.7 Realisieren

2.7.1 Typo3 Backend

Um die Inline Editing Funktionalität als QFQ-Entwickler zu implementieren, wird wie folgt der "special column name _edit" auf einem QFQ-Content-Element auf einer Typo3 Page definiert. Als Beispiel ein Code-Snippet einer möglichen Anwendung in Form eines QFQ-Blocks der eine Tabelle erzeugt.

```
10 {
    sql = SELECT CONCAT('form=ipa&fe=text1&r=',id) AS _edit
                ,CONCAT('table=Dummy&column=text2&type=text&r=',id) AS _edit
                ,CONCAT('table=Dummy&column=text3&type=text&r=',id) AS _edit
          FROM Dummy LIMIT 5

    head = <table><thead><tr><th>text1</th><th>text2</th><th>text3</th></tr></thead><tbody>
    tail = </tbody></table>
    rbeg = <tr>
    rend = </td>
    fbeg = <td>
    fend = </td>
}
```

Abbildung 13: Typo3 Backend QFQ-Content-Element

2.7.2 Report.php

Durch die Nutzung der Syntax "AS _edit", wird beim Laden dieser Seite über eine Anfrage an den Server schlussendlich die für mich relevante Klasse "Report.php" aufgerufen, die dafür zuständig ist "QFQ-Reports" aufzubauen und auszuliefern. Ohne unnötig ins Detail zu gehen werden alle Spalten, die in einem QFQ-Block definiert wurden, mit der Funktion "renderColumn()" entgegen genommen.

Dabei wichtig für mein Feature sind die Variablen: "$columnValue" und "$columnName".

"$columnName" beinhaltet den "special column name" "_edit" als String und wird in einem Switch Case Statement mit allen verschiedenen Konstanten die für die entsprechenden "special column names" definiert wurden abgeglichen bis schliesslich die unten sichtbare Codestelle erreicht wird.

Die Variabel "$columnValue" beinhaltet alle angegebenen Werte innerhalb der "CONCAT()" Funktion, wie in Abbildung 13 ersichtlich, als String. Diese werden der Funktion "process()" der InlineEdit Instanz übergeben.

```
        case COLUMN_INLINE_EDIT:
            // Process given parameters and render inline edit container and label
            $inlineEdit = new InlineEdit($this->dbArr[$this->dbIndexData]);
            $inlineEdit->process($columnValue);
            $content .= $inlineEdit->render();
            break;
```

Abbildung 14: Special Column Name "_edit" Einstieg

2.7.3 process Funktion

Angekommen in der "process()" Funktion werden als erstes die übergebenen Parameter von einem String in einen Array umgewandelt. Die Funktion "KeyValueStringParser::explodeKvpSimple()" teilt den Parameter-String anhand des Gleichheitszeichens (=) und des Ampersands (&) auf und speichert die Ergebnisse im Array $parsedParameters.

```
// Convert parameters from string to array
$parsedParameters = KeyValueStringParser::explodeKvpSimple($parameters, keyValueDelimiter: '=', listDelimiter: '&');
```

Abbildung 15: Parsing mit explodeKvpSimple

Als nächstes wird überprüft, ob die nötigen Parameter angegeben wurden und für welche der beiden Varianten sich entschieden wurde.

2.7.3.1 Variante 1: Referenzierung

```
// Check if 'form', 'fe' and 'r' keys exist
if (array_key_exists( key: INLINE_EDIT_FORM, $parsedParameters)
    && array_key_exists( key: INLINE_EDIT_FORM_ELEMENT, $parsedParameters)
    && array_key_exists( key: INLINE_EDIT_RECORD_ID, $parsedParameters)) {

    // Set properties from parsedParameters
    $this->formName = $parsedParameters[INLINE_EDIT_FORM];

    // Get tableName based on formName
    $sql = SqlQuery::selectFormByName($this->formName);
    $form = $this->database->sql($sql[0], mode: ROW_EXPECT_0_1, $sql[1]);
```

Abbildung 16: Variante 1 Referenz

Der angegebene Form Name wird in der dazugehörigen Property des "InlineEdit" Objekt abgelegt und dann der "selectFormByName()" Funktion übergeben um die Werte für die "sql()" Funktion vorzubereiten.

Die Variable "$sql" wird mit der SQL-Abfrage und den Parametern gefüllt, die im anschließenden "Prepared Statement" übergeben werden, das durch die Funktion "sql()" ausgelöst wird..

```
$sql = {array[2]}
  0 = "SELECT * FROM `Form` AS f WHERE `f`.`name` LIKE ? AND `f`.`deleted`='no'"
  1 = {array[1]}
    0 = "ipa"
```

Abbildung 17: Aufbau für Prepared Statement

Die "sql()" Funktion gibt, wenn ein existierender Form Name im Typo3 Backend angegeben wurde, einen Array zurück mit allen Information über das Form. Wichtig zu wissen ist, dass bei der Definition eines QFQ-Forms eindeutigen Namen vergeben wird. Den einzigen Wert der momentan gebraucht wird ist das Element des Arrays mit dem Key "tableName".

Wenn ein Formularname angegeben wurde, der nicht existiert, gibt die Funktion "sql()" ein leeres Array zurück. Daher würde in der folgenden "if" Bedingung eine "UserReportException" ausgegeben und somit der weitere Ablauf der Funktion an dieser Stelle abgebrochen. Ist der Formname jedoch korrekt, können die später benötigten Properties gesetzt werden.

```
// Check if sql query found a form
if(!isset($form[INLINE_EDIT_TABLE_NAME])){
    throw new \UserReportException ( message: 'Form "'. $parsedParameters[INLINE_EDIT_FORM] .'" does not exist.'
        , code: ERROR_INLINE_EDIT_FORM_NONEXISTENT
    );
}

// Set properties from parsedParameters
$this->formId = $form[F_ID];
$this->table = $form[INLINE_EDIT_TABLE_NAME];
$this->column = $parsedParameters[INLINE_EDIT_FORM_ELEMENT];
$this->recordId = $parsedParameters[INLINE_EDIT_RECORD_ID];
```

Abbildung 18: Fehlendes Form Fehlerbehandlung

Die letzte Information, die uns für diese Variante noch fehlt, ist der Typ des "FormElements", dessen Name angegeben wurde. Zunächst muss man wissen, dass jedes "FormElement" Name pro zugehörigem Form eindeutig sein muss. Außerdem entspricht der "FormElement" Name einer Spalte der Tabelle, die in der Form angegeben wurde. Wir wählen also das "FormElement" über seinen Namen und den Namen des zugehörigen Formulars aus.

```
// Get formElement to get its type based on column and formName
$sql = SqlQuery::selectFormElementByName($this->column, $this->formName);
$formElement = $this->database->sql($sql[0], mode: ROW_EXPECT_0_1, $sql[1]);

// Check if sql query found a formElement
if(!isset($formElement[FE_TYPE])){
    throw new \UserReportException ( message: 'FormElement "'
                                    . $parsedParameters[INLINE_EDIT_FORM_ELEMENT]
                                    .'" does not exist on given Form "'
                                    . $this->formName .'".'
        , code: ERROR_INLINE_EDIT_FORM_ELEMENT_NONEXISTENT
    );
}

$this->feType = $formElement[FE_TYPE];
```

Abbildung 19: Fehlendes Form Element Fehlerbehandlung

Wenn das "FormElement" nicht gefunden werden kann, wird der weitere Funktionsablauf unterbrochen und es wird dem User eine aussagekräftige Fehlermeldung angezeigt. Wenn aber das "FormElement" gefunden wurde, kann die "feType" Property gesetzt werden.

2.7.3.2 Variante 2: Direkt

Vorausgegeben es wurden alle notwendigen Parameter im Typo3 Backend definiert, können in der zweiten Variante die Properties der "InlineEdit" Instanz direkt gesetzt werden.

```
// Check if 'table', 'column', 'type' and 'r' keys exist
} elseif (array_key_exists( key: INLINE_EDIT_TABLE, $parsedParameters)
    && array_key_exists( key: INLINE_EDIT_COLUMN, $parsedParameters)
    && array_key_exists( key: FE_TYPE, $parsedParameters)
    && array_key_exists( key: INLINE_EDIT_RECORD_ID, $parsedParameters)) {

    // Set properties with the values from the parsed parameters
    $this->table = $parsedParameters[INLINE_EDIT_TABLE];
    $this->column = $parsedParameters[INLINE_EDIT_COLUMN];
    $this->feType = $parsedParameters[FE_TYPE];
    $this->recordId = $parsedParameters[INLINE_EDIT_RECORD_ID];
```

Abbildung 20: Variante 2 Direkt

2.7.3.3 Fehlende oder invalide Parameter

Für den Fall, dass keiner der beiden Varianten alle notwendigen Parameter erhalten hat, oder ein ungültiger Parameter angegeben wurde. Habe ich die Funktion "checkGivenParameters()" geschrieben.

```
    // Error handling missing or invalid parameters
} else {
    $error = $this->checkGivenParameters($parsedParameters);
    throw new \UserReportException ($error[INLINE_EDIT_ERROR_MESSAGE], $error[INLINE_EDIT_ERROR_CODE]);
}
```

Abbildung 21: checkGivenParameter Fehlerbehandlung

2.7.4 checkGivenParameters Funktion

Die Funktion "checkGivenParameters()" überprüft die gegebenen Parameter auf Gültigkeit und Vollständigkeit. Wenn ungültige oder fehlende Parameter übergeben werden, wird eine spezifische und aussagkräftige Fehlermeldung mit einem Fehlercode zurückgegeben.

```php
private function checkGivenParameters(array $parsedParameters): array {
    if (isset($parsedParameters[INLINE_EDIT_FORM])) {
        // Version reference mode: 'form=ipa&fe=text1&r=2'
        $missingParameters = array();
        if (!array_key_exists( key: INLINE_EDIT_FORM_ELEMENT, $parsedParameters)) {
            $missingParameters[] = INLINE_EDIT_FORM_ELEMENT;
        }
        if (!array_key_exists( key: INLINE_EDIT_RECORD_ID, $parsedParameters)) {
            $missingParameters[] = INLINE_EDIT_RECORD_ID;
        }
        $errorMessage = 'Missing parameters for special column name "_edit" : ' . implode( separator: ', ', $missingParameters);
        $errorCode = ERROR_INLINE_EDIT_MISSING_REQUIRED_PARAMETER;
    } elseif (isset($parsedParameters[INLINE_EDIT_TABLE])) {
        // Version direct mode: 'table=Dummy&column=text1&type=text&r=2'
        $missingParameters = array();
        if (!array_key_exists( key: INLINE_EDIT_COLUMN, $parsedParameters)) {
            $missingParameters[] = INLINE_EDIT_COLUMN;
        }
        if (!array_key_exists( key: FE_TYPE, $parsedParameters)) {
            $missingParameters[] = FE_TYPE;
        }
        if (!array_key_exists( key: INLINE_EDIT_RECORD_ID, $parsedParameters)) {
            $missingParameters[] = INLINE_EDIT_RECORD_ID;
        }
        $errorMessage = 'Missing parameters for special column name "_edit": ' . implode( separator: ', ', $missingParameters);
        $errorCode = ERROR_INLINE_EDIT_MISSING_REQUIRED_PARAMETER;
    } else {
        $errorMessage = 'Invalid parameters for special column name "_edit" provided.';
        $errorCode = ERROR_INLINE_EDIT_INVALID_PARAMETER_GIVEN;
    }

    return array(INLINE_EDIT_ERROR_MESSAGE => $errorMessage, INLINE_EDIT_ERROR_CODE => $errorCode);
}
```

Abbildung 22: Funktion checkGivenParameter

Als letztes wird noch überprüft, ob der Datensatz mit der angegebenen "id" überhaupt existiert. Fall dies nicht der Fall ist wird dem User eine Fehlermeldung angezeigt und der Funktionsablauf wird hier unterbrochen.

```php
// Get value given the recordId, tableName and columnName
$result = $this->database->sql( sql: "SELECT $this->column FROM $this->table WHERE `id` = ?", mode: ROW_EXPECT_0_1, [$this->recordId]);

// Check if record for given table exists
if(empty($result)){
    throw new \UserReportException ( message: 'Record with id "'. $parsedParameters[INLINE_EDIT_RECORD_ID]
                                .'" for table "'. $this->table .'" does not exist.', code: ERROR_INLINE_EDIT_RECORD_NONEXISTENT);
}

// Set value to result of query given the columnName
$this->value = $result[$this->column];
```

Abbildung 23: Datensatz existiert nicht Fehlerbehandlung

Falls der Datensatz existiert, kann mit Hilfe des angegeben Spaltennamens das Property "value" von der "InlineEdit" Instanz gesetzt werden. Somit sind wird am Ende der Process() Funktion angekommen. Zusammenfassend wurden die angegebenen Parameter verarbeitet und den entsprechenden Properties des "InlineEdit"-Objekts zugeordnet.

2.7.5 render Funktion

Wir befinden uns weiterhin in der Klasse "Report.php" im "CASE INLINE_EDIT_LOAD". Der nächste Schritt besteht darin, aus den zuvor gefüllten Properties der Instanz "InlineEdit" den HTML-Code für das Eingabefeld zu generieren, den wir dann an den Client ausliefern können.

```php
public function render(): string {
    // &table has to be named 'table' because of QuickFormQuery.php line 517
    // Save values in SIP_STORE, so they later can be used in different instance
    $dataSip = $this->link->renderLink( str: 'p:&r='. $this->recordId
        .'&table='.$this->table
        .'&column='.$this->column
        .'&feType='.$this->feType
        .'&value='. $this->value
        .'&pageId='. ($this->pageId ?? (strval($this->store->getVar( key: TYPO3_PAGE_ID, useStores: STORE_TYPO3))))
        .'&formName='. ($this->formName ?? 'noForm')
        .'&formId='.strval($this->formId)
        .'|s|r:8'
    );

    // Wrap the value with the HTML tags for the label and the inline edit container, sip is stored in the 'data-sip'
    $html = Support::wrapTag( tag: '<div class="qfq-inline-edit-label">',$this->value);
    $html = Support::wrapTag( tag: '<div class="qfq-inline-edit" data-sip="' . $dataSip . '">', $html);

    return $html;
}
```

Abbildung 24: InlineEdit.php render Funktion

Als erstes rufen wir über, dass im "Contructor" initialisierte "Link" Objekt, die Methode "renderLink()" auf. Die ermöglicht es, die für uns wichtigen, Variablen im "Sip Store" abzulegen. Als Rückgabewert wird ein "Sip Key" erzeugt der später im HTML-Attribut "data-sip" abgelegt wird. Somit können wir später auf Variablen im "Sip Store" zugreifen. Für den HTML-Code den wir ausliefern bauen wir zuerst ein Element mit der Klasse "qfq-inline-edit.label" auf. Innerhalb dieses Elementes befindet sich der Wert, den wir anzeigen möchten. Dann umschliessen wir diese Label mit einem anderem Element, dass den Container bildet. Wir vergeben ihm die Klasse "qfq-inline-edit" und definieren wie bereits erwähnt das Attribut "data-sip".

2.7.6 Frontend Ansicht

Der generierte HTML-Code wird nun an den Benutzer ausgeliefert. Gehen wir davon aus, dass der Code wie in der Abbildung 13 im Typo3-Backend definiert wurde und die entsprechenden Datensätze etc. vorhanden sind. würde dem Benutzer folgender Seiteninhalt angezeigt.

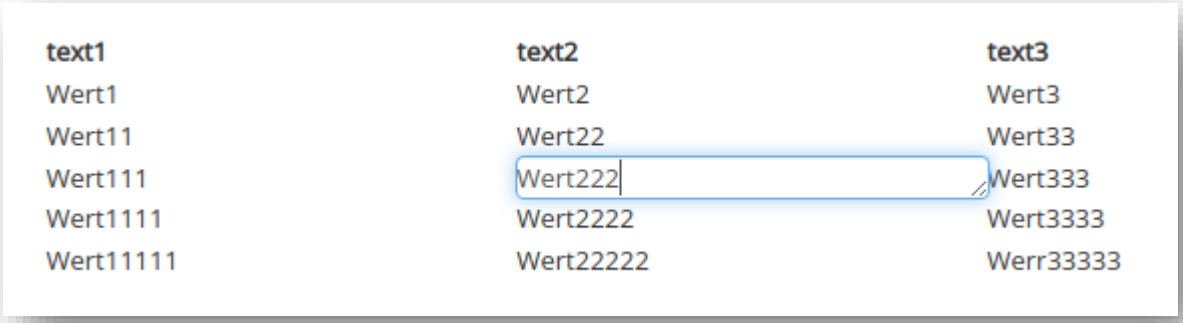
text1	text2	text3
Wert1	Wert2	Wert3
Wert11	Wert22	Wert33
Wert111	Wert222	Wert333
Wert1111	Wert2222	Wert3333
Wert11111	Wert22222	Werr33333

Abbildung 25: Beispiel Inline Edit Tabelle 1

2.7.7 Generierung des Textarea Elements

Wenn der Benutzer nun auf eines der Labels klickt, wird eine "Textarea" erzeugt, die den Wert des Labels enthält und darauf wartet, bearbeitet zu werden. Eine einfachere Lösung wäre gewesen, für jedes Label die entsprechende "Textarea" beim Laden der Seite zu liefern. Je nach Anzahl der generierten Inline-Edit-Elemente könnte dies jedoch zu erheblichen Performance-Einbußen führen. Aus diesem Grund haben wir uns entschieden, nur dann eine "Textarea" zu erzeugen, wenn diese auch benötigt wird.

Bevor wir uns anschauen, wie das genau funktioniert, hier ein Screenshot, wie das Ganze aussieht.



text1	text2	text3
Wert1	Wert2	Wert3
Wert11	Wert22	Wert33
Wert111	Wert222	Wert333
Wert1111	Wert2222	Wert3333
Wert11111	Wert22222	Werr33333

Abbildung 26: Beispiel Inline Edit Tabelle 2

2.7.8 JavaScript OnClick Event

Da wir bei der Auslieferung des HTML-Inhalts beim Laden der Seite dem Container die Klasse "qfq-inline-edit" zugewiesen haben, können wir nun hier im JavaScript für alle Elemente dieser Klasse folgende Funktion definieren. Zuerst wird die Variable "dataSip" mit dem Wert des Attributs "dataSip" gefüllt, dann werden zwei JQUERY-Objekte erzeugt, das "Textarea"-Element, falls bereits eines vorhanden ist und das Label, das auf jeden Fall vorhanden ist. Falls noch keine "Textarea" existiert, wird die Funktion "retrieveTextbox()" aufgerufen, der die "dataSip" und das Label übergeben werden. Falls bereits eine "Textarea" erzeugt wurde, kann dieser die Klasse "hidden" entzogen und dem Label hinzugefügt werden.

```
$(document).ready( fn: function() : void {
    (function(n : {} ) : void {
        $('.qfq-inline-edit').on('click', function() : void {
            var dataSip = $(this).data('sip');
            var $input = $(this).children('textarea');
            var $label = $(this).children('div.qfq-inline-edit-label');

            if ($input.length === 0) {
                // If the textarea doesn't exist, retrieve it
                retrieveTextbox(dataSip, $label);
            } else {
                $input.removeClass('hidden');
                $input.focus();
                $label.addClass('hidden');
            }
        });
    })(QfqNS);
});
```

Abbildung 27: OnClick Event

2.7.9 retrieveTextbox Funktion

In der Funktion wird eine Ajax-Anfrage an die URL "typo3conf/ext/qfq/Classes/Api/inlineEditLoad.php?s=" + "dataSip" gesendet. Diese Anfrage erfolgt über die POST-Methode. Bei erfolgreicher Anfrage wird die Funktion "success" ausgeführt, andernfalls wird die Funktion "error" aufgerufen.

Im "success"-Block wird die erhaltenen Daten (response) verwendet, um eine "Textarea" zu erstellen und deren Eigenschaften anzupassen. Zudem werden verschiedene Funktionen aufgerufen, um das Styling anzupassen:

```
function retrieveTextbox(dataSip, $label) :void  {
    $.ajax({
        url: "typo3conf/ext/qfq/Classes/Api/inlineEditLoad.php?s=" + dataSip,
        method: "POST",
        success: function(response) :void  {
            // Create the textbox and modify its properties
            var $textBox = createTextbox(response);
            $label.addClass('hidden');
            insertAfterLabel($label, $textBox);
            setTextboxHeight($textBox);
            enableAutoGrow($textBox);
            adjustContainerHeight($textBox, $label, $mode: 'textBox');
            registerBlurHandler($textBox, dataSip, $label);
        },
        error: function() :void  {
            console.error('inlineEditLoad.php API error');
        }
    });
}
```

Abbildung 28: retrieveTextbox Funktion

- createTextbox(response) erstellt die "Textarea".
- insertAfterLabel($label, $textBox) fügt die "Textarea" nach dem Label ein.
- setTextboxHeight($textBox) setzt die Höhe der "Textarea".
- enableAutoGrow($textBox) aktiviert die automatische Anpassung der "Textarea"-Größe.
- adjustContainerHeight($textBox, $label, 'textBox') passt die Höhe des Containers basierend auf der "Textarea" an.
- registerBlurHandler($textBox, dataSip, $label) registriert einen Event-Handler für das Auslösen eines Blur-Ereignisses auf der "Textarea".

Im "error"-Block wird eine Fehlermeldung in der Konsole ausgegeben, falls ein Fehler bei der API-Anfrage auftritt.

2.7.10 inlineEditLoad.php API

Es wird ein Versuch (try-catch) gestartet, um mögliche Fehler abzufangen.

Innerhalb des Versuchs wird ein neues Objekt der Klasse "QuickFormQuery" erstellt und die "inlineEdit" Funktion mit dem Parameter "FORM_INLINE_EDIT_LOAD" aufgerufen. Das Ergebnis wird dem "$answer"-Array zugewiesen. Diese Funktion generiert eine HTML-Repräsentation des "Textarea" Elements.

Wenn eine "UserFormException" oder "CodeException" auftritt, werden die entsprechenden Fehlernachrichten in das "$answer"-Array geschrieben.

Der Content-Type der Antwort wird auf "application/json" festgelegt, da die Antwort im JSON-Format an den aufrufenden Client zurückgegeben wird.

Die Antwort wird mit Hilfe der Funktion" json_encode" in das JSON-Format umgewandelt und an den aufrufenden Client zurückgegeben.

```php
try {
    try {

        // Needed to access STORE_SIP and generate formElement
        $qfq = new QuickFormQuery(['bodytext' => '']);

        // Return HTML representation of formElement to js
        $answer = $qfq->inlineEdit( mode: FORM_INLINE_EDIT_LOAD);

    } catch (\UserFormException $e) {
        $answer[API_MESSAGE] = $e->formatMessage();
    } catch (\CodeException $e) {
        $answer[API_MESSAGE] = $e->formatMessage();
    }
} catch (\Throwable $e) {
    $answer[API_MESSAGE] = "Generic Exception: " . $e->getMessage();
}

header( header: "Content-Type: application/json");

echo json_encode($answer);
```

Abbildung 29: inlineEditLoad.php API

2.7.11 case FORM_INLINE_EDIT_LOAD

Über die "inlineEdit()" Funktion wird die private Funktion "doForm()" aufgerufen, zudem wird ihr die konstante "FORM_INLINE_EDIT_LOAD" übergeben. Folgender Code wird ausgeführt:

```
case FORM_INLINE_EDIT_LOAD:
    $renderer = new FormElementRenderBS3();
    // Create formElement based on variables stored in STORE_SIP
    $fe = FeFactory::createFe($this->store::getVar( key: INLINE_EDIT_RECORD_ID, useStores: STORE_SIP),
        $this->store::getVar( key: FE_VALUE, useStores: STORE_SIP),
        $this->store::getVar( key: INLINE_EDIT_COLUMN, useStores: STORE_SIP),
        $this->store::getVar( key: INLINE_EDIT_TABLE, useStores: STORE_SIP),
        $this->store::getVar( key: INLINE_EDIT_FE_TYPE, useStores: STORE_SIP),
        $this->dbIndexData
    );
    // Prepare feAttributes for rendering
    $fe->process();
    // Generate HTML representation of formElement
    $data = $renderer->renderFormElement($fe);
    break;
```

Abbildung 30: doForm FORM_INLINE_EDIT_LOAD case

Es wird mithilfe der statischen Funktion "createFe" der "FeFactory" Klasse ein Objekt der Klasse "FormElementInput" erstellt.

2.7.12 createFE Funktion

Die Funktion "createFe" erstellt und gibt eine Instanz einer konkreten Klasse zurück, die von der abstrakten Klasse "AbstractFormElement" erbt. Die konkrete Klasse wird basierend auf dem übergebenen "feType" (Formelement-Typ) erstellt.

```
public static function createFe(string $recordId, string $value, string $column
    , string $table, string $feType, string $dbIndex):AbstractFormElement {

    // Define needed default values
    $formElement = array(FE_NAME => $column,
        FE_MODE_SQL => '',
        FE_ENCODE => 'specialchar',
        FE_CHECK_TYPE => 'auto',
        FE_MODE => 'show',
        FE_HTML_ID => $column . '-' . $table . '-' . $recordId,
    );

    $htmlFormElementName = $column . '-' . $table . '-' . $recordId;

    // Design Pattern: Factory
    // Instantiate the formElement object based on the feType
    switch ($feType){
        case FE_TYPE_TEXT:
            $formElement[FE_TYPE] = 'text';
            $formElement[FE_SIZE] = '1,0,1';

            #todo would ne nice to not call setFeDefault() for every type, only once would be better
            $formElement = Support::setFeDefaults($formElement);

            $fe = new FormElementInput($formElement, $value, $htmlFormElementName, $dbIndex);
            break;

        case FE_TYPE_SELECT:...

        case FE_TYPE_UPLOAD:...

    }

    return $fe;

}
```

Abbildung 31: createFe Funktion

Die Funktion verwendet ein assoziatives Array "$formElement", um die benötigten Standardwerte für das "FormularElement" zu definieren. Es enthält Eigenschaften wie "name", "modeSql", "encode", "checkType", "mode" und "FE_HTML_ID". Der Wert für "FE_HTML_ID" wird basierend auf der Kombination von "$column", "$table" und "$recordId" generiert.

Die Funktion implementiert das Factory Design Pattern. Sie instanziiert das "FormElement"-Objekt basierend auf dem Wert von "$feType". Dazu verwendet sie einen " switch"-Block, der den " $feType" überprüft und entsprechend den passenden "FormularElement"-Typ festlegt.

Für den "Formelement-Typ" "text" wird ein "FormElementInput"-Objekt erstellt, wobei die Eigenschaften aus "$formElement" übergeben werden. Das erstellte Objekt wird mit "$value", "$htmlFormElementName" und "$dbIndex" initialisiert.

Für die anderen geplanten "FormElement"-Typen "select", "upload", "date", "time" und "datetime" wird der entsprechende "FormElement"-Typ im "$formElement"-Array gesetzt.

Am Ende wird, das erstellte "FormElement"-Objekt zurückgegeben.

2.7.13 process Funktion und renderFormElement Funktion

Anschließend wird das Objekt mit der "process"-Funktion von "FormElementInput" für das Rendering vorbereitet. Zum Beispiel werden die HTML-Attribute gesammelt, die in der Funktion "renderFormElement" benötigt werden.

Die "renderFormElement" Funktion erzeugt die HTML-Repräsentation des "FormElement", die wir nun über die "doForm" Funktion des "QuickFormQuery" Objekts der "inlineEditLoad.php" API zurückgeben können. Diese wiederum liefert den generierten HTML-Code als Antwort auf das "onClick"-Ereignis, das alles ausgelöst hat.

2.7.14 createTextbox Funktion

Zurück im JavaScript wird nun die "Response" des Ajax-Calls an die Funktion "createTextbox()" übergeben. Diese erzeugt dann ein "Textarea"-Element, das nach dem Label im selben Container eingefügt werden kann. Damit wird das "Textarea"-Element für den Benutzer sichtbar. Zusätzlich wird dem "Textarea" die Klasse "qfq-inline-edit-input" übergeben.

```
function createTextbox(response) {
    // Create a textbox element from the server response
    var $textBox = $(response);
    $textBox.addClass('qfq-inline-edit-input');
    $textBox.attr('spellcheck', 'false');
    return $textBox;
}
```

Abbildung 32: createTextbox Funktion

2.7.15 registerBlurHandler Funktion

Die Funktion "registerBlurHandler" registriert den Blur-Ereignishandler für die "Textarea" ("$textBox"). Wenn die "Textarea" den Fokus verliert (Blur-Ereignis), wird der Ereignishandler ausgelöst.

Innerhalb des Ereignishandlers wird der aktualisierte Wert der "Textarea" abgerufen ("updatedValue). Anschließend wird eine "Ajax"-Anfrage gesendet, um den Datensatz in der Datenbank zu aktualisieren. Dazu wird die URL "typo3conf/ext/qfq/Classes/Api/inlineEditSave.php?s=" + "dataSip" verwendet, wobei "dataSip" ein Parameter ist, der an die Funktion übergeben wird.

Die "Ajax"-Anfrage wird als "POST"-Anfrage gesendet und übermittelt den aktualisierten Wert ("updatedValue") als "data". Bei erfolgreichem Abschluss der Anfrage wird die "success"-Funktion aufgerufen. In dieser Funktion werden folgende Aktionen durchgeführt:

- Die "Textarea" ("$textBox") wird mit der CSS-Klasse "hidden" versehen, um sie auszublenden.
- Der Text des Labels ("$label") wird auf die vom Server zurückgegebene Antwort ("response") gesetzt, und die CSS-Klasse "hidden" wird entfernt, um das Label anzuzeigen.
- Die Funktion "adjustContainerHeight" wird aufgerufen, um die Höhe des Containers anzupassen, der das "Textarea" und das Label enthält.

Wenn bei der "Ajax"-Anfrage ein Fehler auftritt, wird die "error"-Funktion aufgerufen. In dieser Funktion werden folgende Aktionen durchgeführt:

- Die "Textarea" ("$textBox") wird mit der CSS-Klasse "hidden" versehen, um es auszublenden.
- Die CSS-Klasse "hidden" wird vom Label ("$label") entfernt, um es anzuzeigen.

```
function registerBlurHandler($textBox, dataSip, $label) : void  {
    // Register the blur event handler for the textbox
    $textBox.on('blur', function() : void  {
        var updatedValue = $(this).val();

        // Send an AJAX request to update the record in the database
        $.ajax({
            url: "typo3conf/ext/qfq/Classes/Api/inlineEditSave.php?s=" + dataSip,
            method: "POST",
            data: { 'updatedValue': updatedValue },
            success: function(response) : void  {
                $textBox.addClass('hidden');
                $label.text(response).removeClass('hidden');
                adjustContainerHeight($textBox, $label, $mode: 'label');
            },
            error: function() : void  {
                $textBox.addClass('hidden');
                $label.removeClass('hidden');
            }
        });
    });
}
```

Abbildung 33: registerBlurHandler Funktion

2.7.16 case FORM_INLINE_EDIT_SAVE

Die "inlineEditSave.php" API ruft ebenfalls über die "inlineEdit" Funktion die private Funktion "doForm" auf und übergibt ihr diesmal den Mode "FORM_INLINE_EDIT_SAVE".

```
case FORM_INLINE_EDIT_SAVE:
    // Get all necessary values for the update
    $table = $this->store->getVar( key: INLINE_EDIT_TABLE, useStores: STORE_SIP);
    $column = $this->store->getVar( key: INLINE_EDIT_COLUMN, useStores: STORE_SIP);
    $recordId = $this->store->getVar( key: INLINE_EDIT_RECORD_ID, useStores: STORE_SIP);
    $updatedValue = $this->store->getVar( key: INLINE_EDIT_UPDATED_VALUE, useStores: STORE_CLIENT . STORE_ZERO
        , sanitizeClass: SANITIZE_ALLOW_ALL);
    $oldValue = $this->store->getVar( key: FE_VALUE, useStores: STORE_SIP, sanitizeClass: SANITIZE_ALLOW_ALL);
    // Check if changes were made
    if($updatedValue !== $oldValue){
        // Update the record with the new value
        $this->dbArray[$this->dbIndexQfq]->sql( sql: "UPDATE $table SET $column = ? WHERE `id` = ?"
            , mode: ROW_EXPECT_1,[$updatedValue,$recordId]);
        $data = $updatedValue;
        // Prepare formSpec for logging
        $this->formSpec[F_DO_NOT_LOG_COLUMN] = '';
        $this->formSpec[F_ID] = $this->store->getVar( key: FE_FORM_ID, useStores: STORE_SIP) ?? '0';
        $this->formSpec[F_NAME] = $this->store->getVar( key: INLINE_EDIT_FORM_NAME, useStores: STORE_SIP);
        // Log to formSubmitLog table
        $this->logFormSubmitRequest( mode: FORM_INLINE_EDIT_SAVE);
    } else {
        $data = $oldValue;
    }
    break;
```

Abbildung 34: doForm case FORM_INLINE_EDIT_SAVE

Als Erstes werden verschiedene Werte aus dem Client und Sip Store abgerufen, die für das Update benötigt werden, einschließlich des Tabellennamens ("$table"), des Spaltennamens ("$column"), der Datensatz-ID ("$recordId"), des aktualisierten Werts ("$updatedValue") und des alten Werts ("$oldValue"). Es wird überprüft, ob Änderungen vorgenommen wurden, indem der aktualisierte Wert ("$updatedValue") mit dem alten Wert ("$oldValue") verglichen wird. Falls Änderungen vorgenommen wurden (d.h. "$updatedValue" ist nicht gleich "$oldValue"), wird der Datensatz in der Datenbank mit dem neuen Wert aktualisiert. Dazu wird ein "SQL-Update-Prepared-Statement" ausgeführt, dass den Tabellennamen, den Spaltennamen und die Datensatz-ID verwendet.

Der aktualisierte Wert ("$updatedValue") wird der Variablen "$data" zugewiesen.

Das "$formSpec"-Array wird eingerichtet, um es für das Logging vorzubereiten. Einige Werte werden gesetzt, wie z.B. die Spalte für das Logging ausschließen ("F_DO_NOT_LOG_COLUMN"), die Formular-ID ("F_ID") und der Formularname ("F_NAME").

Die Funktion "logFormSubmitRequest" wird aufgerufen, um das Ereignis in die Tabelle "FormSubmitLog" zu protokollieren. Der Parameter "FORM_INLINE_EDIT_LOAD" wird übergeben um sicherzustellen, dass der erzeugte Log Datensatz dem Inline Edit Feature zugewiesen werden kann.

Falls keine Änderungen vorgenommen wurden, wird der alte Wert ("$oldValue") der Variablen "$data" zugewiesen.

Die Variabel "$data" wird der API und dann dem JavaScript geliefert, dass schlussendlich der Wert des Labels mit dem über die "Textarea" gesetzten wert gesetzt werden kann.

2.7.17 Fehlerbehandlung

Die Fehlerbehandlung ist wichtig, um unerwartete Situationen oder ungültige Eingaben abzufangen und dem Benutzer oder Entwickler eine angemessene Rückmeldung zu geben. Durch die Verwendung von Ausnahmen können spezifische Fehlermeldungen und Fehlercodes generiert werden, die den Fehler genau beschreiben und eine gezielte Fehlerbehebung ermöglichen.

In der QFQ-Codebase werden hauptsächlich die folgenden vier Arten von Exceptions gebraucht:

1. UserFormException
2. UserReportException
3. CodeException
4. DbException

2.7.17.1 Beispiele

Bei der Angabe von: **10.sql = SELECT CONCAT('table=Dummy&r=',id) AS _edit FROM Dummy**, wird dem User angezeigt, dass die Parameter column und type fehlen.

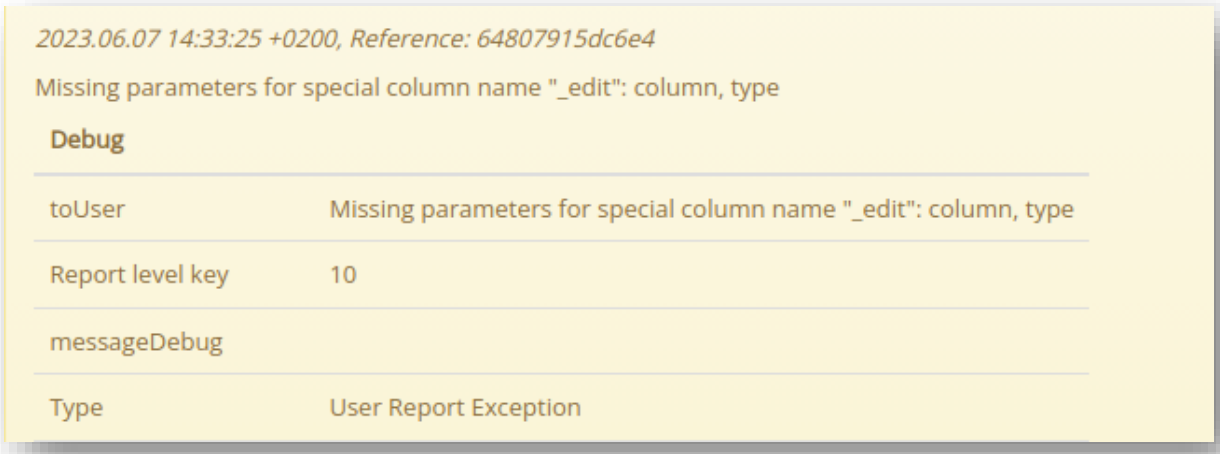


Abbildung 35: Fehlermeldung missing parameters

Oder bei der Angabe von **10.sql = SELECT 'table=Dummy&column=text1&type=text&r=999' AS _edit FROM Dummy**, wird dem User angezeigt folgende Fehlermeldung angezeigt, vorausgesetzt der Datensatz der Tabelle Dummy mit der id 999 existiert nicht.

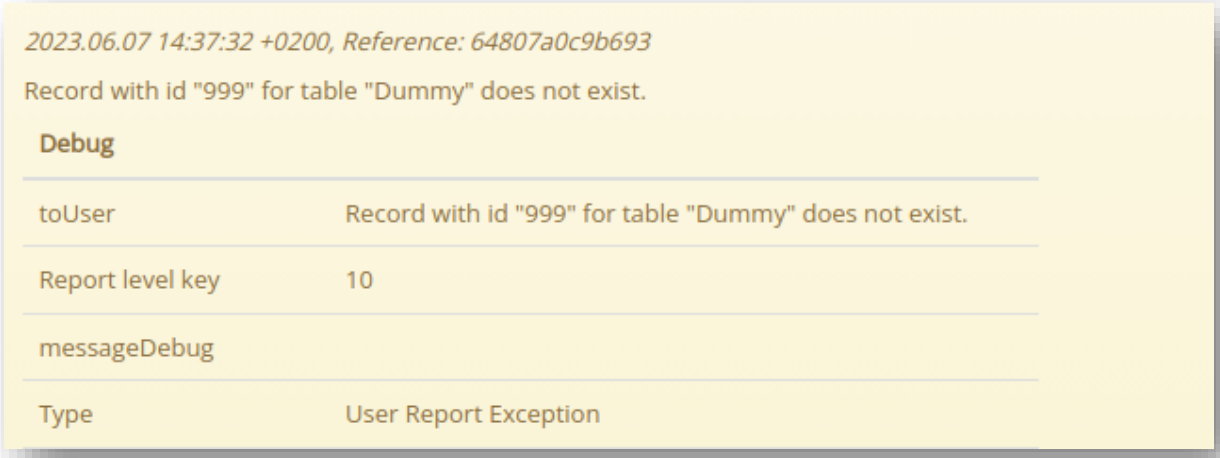


Abbildung 36: Fehlermeldung record does not exist

2.7.18 Logging

Das Logging für mein Inline Edit Feature wird mithilfe der bereits existierenden Funktion "logFormSubmitRequest" implementiert.

```
    $formData = json_encode($formData, flags: JSON_UNESCAPED_UNICODE);

    $sql = "INSERT INTO `FormSubmitLog` (`formData`, `sipData`, `mode`, `clientIp`, `feUser`, `userAgent`
                                        , `formId`, `formName`, `recordId`, `pageId`, `sessionId`, `created`)" .
        "VALUES (?, ?, ?, ?, ?, ?,  ?, ?, ?, ?, ?, NOW())";

    $clientIp = $_SERVER[CLIENT_REMOTE_ADDRESS] ?? '';
    $userAgent = $_SERVER[CLIENT_HTTP_USER_AGENT] ?? '';
    $sipData = json_encode($this->store->getStore( store: STORE_SIP), flags: JSON_UNESCAPED_UNICODE);
    $formId = $this->formSpec[F_ID];
    $formName = $this->formSpec[F_NAME];
    $feUser = $this->store->getVar( key: TYPO3_FE_USER, useStores: STORE_TYPO3, sanitizeClass: SANITIZE_ALLOW_ALNUMX);
    $pageId = $this->store->getVar( key: TYPO3_PAGE_ID, useStores: STORE_TYPO3, sanitizeClass: SANITIZE_ALLOW_ALNUMX);
    // needed to log pageId in case of inline editing
    if(!$pageId){
        $pageId = $this->store->getVar( key: TYPO3_PAGE_ID, useStores: STORE_SIP, sanitizeClass: SANITIZE_ALLOW_ALNUMX);
    }
    $sessionId = session_id();

    $params = [$formData, $sipData, $mode, $clientIp, $feUser, $userAgent, $formId, $formName, $recordId, $pageId, $sessionId];

    $this->dbArray[$this->dbIndexQfq]->sql($sql, mode: ROW_REGULAR, $params);
}
```

Abbildung 37: logFormSubmitRequest Funktion

Die Funktion "logFormSubmitRequest" dient dazu, Änderungen von Werten die über ein "QFQ-Form" gemacht wurden in einer Datenbanktabelle namens "FormSubmitLog" zu protokollieren.

- Wenn ein eingeloggter Frontend-Benutzer vorhanden ist, wird sein Benutzername in die Spalte **"feUser**" eingetragen.
- Der Timestamp **"created"** speichert das Datum und die Uhrzeit, an dem die Änderung vorgenommen wurde.
- Die Spalte **"mode"** dient dazu, die Änderungen, die durch das Inline-Editing-Feature vorgenommen wurden, von denen zu unterscheiden, die über ein QFQ-Form gemacht wurden. Entweder "form_inline_edit_save" oder "form_save".
- Im Log des Inline-Editing-Features sind der neue Wert **"updatedValue"** in der Spalte **"formData"** und der alte Wert in der Spalte **"sipData"** ersichtlich.
- Die Spalte **"formId"** wird bei der direkten Variante vom Inline Editing Feature immer auf 0 gesetzt, da sie nicht von einem Form abhängig ist. Ansonsten beinhaltet sie die id des Formulars.
- In der Spalte **"formName**" wird der Name des QFQ-Forms abgelegt, falls keines existiert, wird der String "noForm" gespeichert.
- Zudem wird die Typo3 **"pageId"** gespeichert.

2.7.19 FormSubmitLog Tabelle

Die Datei "qfqDefaultTables.sql" generiert alle Tabellen, die mit QFQ ausgeliefert werden.

Dieses Skript wird entweder automatisch bei der ersten Installation von QFQ ausgeführt oder kann über einen Parameter bei jedem Laden einer Seite mit aktivierter QFQ-Erweiterung ausgeführt werden. Die "FormSubmitLog" Tabelle ist in diesem Skript enthalten.

```
CREATE TABLE IF NOT EXISTS `FormSubmitLog`
(
    `id`        INT(11)      NOT NULL AUTO_INCREMENT,
    `formData`  TEXT         NOT NULL,
    `sipData`   TEXT         NOT NULL,
    `mode`      VARCHAR(32)  NOT NULL,
    `clientIp`  VARCHAR(64)  NOT NULL,
    `feUser`    VARCHAR(64)  NOT NULL,
    `userAgent` TEXT         NOT NULL,
    `formId`    INT(11)      NOT NULL DEFAULT '0',
    `formName`  VARCHAR(255) NOT NULL,
    `recordId`  INT(11)      NOT NULL DEFAULT '0',
    `pageId`    INT          NOT NULL,
    `sessionId` VARCHAR(32)  NOT NULL,
    `created`   TIMESTAMP    NOT NULL DEFAULT CURRENT_TIMESTAMP,

    PRIMARY KEY (`id`),
    INDEX (`feUser`),
    INDEX (`clientIp`),
    INDEX (`feUser`),
    INDEX (`mode`),
    INDEX (`formId`),
    INDEX (`formName`),
    INDEX (`recordId`),
    INDEX (`pageId`)
)
    ENGINE = InnoDB
    DEFAULT CHARSET = utf8
    AUTO_INCREMENT = 0;

ALTER TABLE `FormSubmitLog` ADD COLUMN IF NOT EXISTS `mode` varchar(32) NOT NULL AFTER sipData;
```

Abbildung 38: Default Table FormSubmitLog

Um nachvollziehen zu können, von wo aus (Form oder Inline Editing) die Änderung vorgenommen wurde.

Habe ich der Tabelle "FormSubmitLog" eine Spalte namens "mode" hinzugefügt.

Außerdem habe ich das "ALTER TABLE ADD COLUMN IF NOT EXIST" Statement geschrieben, um sicherzustellen, dass alle QFQ Installationen, die bereits die "FormSubmitLog" Tabelle in ihrer Datenbank haben, die neue Spalte "mode" hinzugefügt bekommen.

Dieser Datensatz ist der Log Eintrag von einer Bearbeitung eines Datensatzes über ein QFQ-Form.

Abbildung 39: Beispiel Datensatz 1

Dieser Datensatz ist der Log Eintrag von einer Bearbeitung eines Datensatzes über das Inline Editing Feature mit der Variante "Referenz".

Abbildung 40: Beispiel Datensatz 2

Dieser Datensatz ist der Log Eintrag von einer Bearbeitung eines Datensatzes über das Inline Editing Feature mit der Variante "Direkt".

Abbildung 41: Beispiel Datensatz 3

2.7.20 Benutzeranleitung

Das Benutzerhandbuch "https://docs.qfq.io/en/master/Report.html#column-edit" wird Teil der QFQ-Dokumentation, wenn der Branch dieses Projekts "gemerged" wird. Das Benutzerhandbuch habe ich auf Englisch verfasst, da die gesamte offizielle QFQ-Dokumentation in englischer Sprache verfasst wird.

Column: _edit

Using this special column name you allow the user to directly edit the field of the specified record, without having to open a form. Upon loading the page the value of the specified column is shown, when clicking the label(text), a textarea is generated that lets the page user edit the record. The record will be updated after the textarea loses its focus assuming the value has been changed. All the changes are logged in the 'FormSubmitLog' table, that should automatically exist in your database after installing and using the QFQ extension.

There are two ways this special column name can be used:

1. Direct mode

This variant requires the user to define the following parameters.

- table - table name
- column - column name
- type - input type (currently only 'text' is possible)
- r - record id

```
10.sql = SELECT CONCAT('table=Person&column=firstName&type=text&r=', id) AS _edit FROM Person LIMIT 5
```

2. Reference mode

This variant requires the user to define the following parameters.

- form - form name (table specified in form is used)
- fe - form element name (only form element names that exist as columns on the table specified in the form can be used)
- r - record id

```
10.sql = SELECT CONCAT('form=formName&fe=formElementName&r=', id) AS _edit FROM Person LIMIT 5
```

Example

A possible implementation of the special column name '_edit' could be within a table.

```
10 {
   sql = SELECT CONCAT('table=Dummy&column=text1&type=text&r=',id) AS _edit
               ,CONCAT('table=Dummy&column=text2&type=text&r=',id) AS _edit
               ,CONCAT('table=Dummy&column=text3&type=text&r=',id) AS _edit
         FROM Dummy LIMIT 5

    head = <table><thead><tr><th>text1</th><th>text2</th><th>text3</th></tr></thead><tbody>
    tail = </tbody></table>
    rbeg = <tr>
    rend = </td>
    fbeg = <td>
    fend = </td>
}
```

Limitations

Currently the only compatible input type is 'text'. When using the 'reference' mode, only form element with the type 'text' can be referenced. Additional parameters defined on the referenced form and form element are ignored.

Abbildung 42: Benutzeranleitung QFQ-Doc

2.8 Kontrollieren

2.8.1 Testkonzept

Um sicherzustellen, dass dieses Feature einwandfrei funktioniert, habe ich ein Testkonzept entwickelt. Ich habe mich für die Verwendung von PHP Unit Tests und Blackbox Integrations-Tests entschieden, um die Funktionalität und Benutzererfahrung des Features zu überprüfen.

PHP Unit Tests:

- Testen, ob die neu implementierten Funktionen sich korrekt verhalten.
- Überprüfen, ob die entsprechende Fehlermeldung dem Benutzer angezeigt wird, wenn ungültige Parameter angegeben wurden.

Integrationstest (Black Box):

- Testen, ob das Klicken auf das Label die Generierung einer "Textarea" auslöst.
- Überprüfen, ob das Klicken außerhalb der "Textarea" bewirkt, dass sie verschwindet und das Label sichtbar wird, ohne Änderungen am Wert vorzunehmen (kein Update und kein Log Record).
- Testen, ob das Klicken auf das Label gefolgt von Änderungen im Wert der "Textarea" und anschließendes Verlassen der "Textarea" dazu führt, dass das Label und der Wert in der Datenbank aktualisiert werden.
- Überprüfen, ob alle wichtigen Informationen bei einer Änderung im Log gespeichert werden.

2.8.2 Testumgebung

Beschreibung	Name	Version/Detail
Remote Desktop Client	Thinlinc Client	4.14.0
Betriebssystem/Linux-Distribution	Ubuntu	22.04.2 LTS
Datenbankmanagementsystem	MariaDb	10.8.3
Content Management System	Typo3	10.4.36
Entwicklungsumgebung	PhpStorm	2023.1
Server Side Programmiersprache	PHP	7.4
Webserver	Apache	2.4.54 (Debian)
Browser	Brave	1.52.117

2.8.3 Unit-Tests

Die Klasse InlineEditTest enthält die folgenden Testmethoden:

- **testProcessModeDirect**: Dieser Test überprüft die Verarbeitung des Inline-Editiermodus "Direct". Dabei werden verschiedene Parameter an die "process"-Funktion übergeben und die erwarteten Ergebnisse werden überprüft.
- **testProcessModeReference**: Dieser Test überprüft die Verarbeitung des Inline-Editiermodus "Reference". Dabei werden verschiedene Parameter an die "process"-Funktion übergeben und die erwarteten Ergebnisse werden überprüft.
- **testProcessThrowsExceptionNoForm**: Dieser Test überprüft, ob eine Ausnahme geworfen wird, wenn keine Form gefunden wird. Dabei werden ungültige Formparameter an die "process"-Funktion übergeben und es wird erwartet, dass eine "UserReportException" mit dem entsprechenden Fehlercode geworfen wird.
- **testProcessThrowsExceptionNoFormElement**: Dieser Test überprüft, ob eine Ausnahme geworfen wird, wenn kein "FormElement" gefunden wird. Dabei werden ungültige Formelementparameter an die "process"-Funktion übergeben und es wird erwartet, dass eine "UserReportException" mit dem entsprechenden Fehlercode geworfen wird.
- **testProcessThrowsExceptionNoRecord**: Dieser Test überprüft, ob eine Ausnahme geworfen wird, wenn kein Datensatz gefunden wird. Dabei werden ungültige Datensatzparameter an die "process"-Funktion übergeben und es wird erwartet, dass eine "UserReportException" mit dem entsprechenden Fehlercode geworfen wird.
- **testRender**: Dieser Test überprüft die "Render"-Funktion der "InlineEdit"-Klasse. Dabei wird erwartet, dass der gerenderte HTML-Code den erwarteten Wert hat.
- **testCreateFe**: Dieser Test überprüft die Erstellung eines Formularelements durch die "FeFactory"-Klasse. Dabei wird erwartet, dass ein "FormElementInput"-Objekt zurückgegeben wird.
- **testCheckGivenParametersReferenceMissing**: Dieser Test überprüft, ob eine Ausnahme geworfen wird, wenn fehlende Parameter im Referenzmodus festgestellt werden. Dabei werden ungültige Parameter an die "checkGivenParameters"-Funktion übergeben und es wird erwartet, dass eine entsprechende Fehlermeldung und Fehlercode zurückgegeben werden.
- **testCheckGivenParametersInvalidParameter**: Dieser Test überprüft, ob eine Ausnahme geworfen wird, wenn ungültige Parameter festgestellt werden. Dabei werden ungültige Parameter an die "checkGivenParameters"-Funktion übergeben und es wird erwartet, dass eine entsprechende Fehlermeldung und Fehlercode zurückgegeben werden.

```php
public function testProcessModeReference(){

    // Arrange
    $parameters = 'form=ipa&fe=text1&r=999';
    // Create a mock of the Database class
    $databaseMock = $this->createMock( originalClassName: Database::class);
    // Define the expected behavior of the sql method calls
    $databaseMock->expects($this->at( index: 0))
        ->method('sql')
        ->with(
            "SELECT  *  FROM `Form` AS f WHERE `f`.`name` LIKE ? AND `f`.`deleted`='no'",
            ROW_EXPECT_0_1,
            ['ipa'],
        )
        ->willReturn(['id' => '999', 'tableName' => 'Dummy']);
    $databaseMock->expects($this->at( index: 1))
        ->method('sql')
        ->with(
            "SELECT fe.* FROM `FormElement` AS `fe` JOIN `Form` AS `f` ON `fe`.`formId` = `f`.`id` WHERE `fe`.`name` = ? AND `f`.`name` = ?",
            ROW_EXPECT_0_1,
            [0 => 'text1', 1 => 'ipa'])
        ->willReturn(['type' => 'text']);
    $databaseMock->expects($this->at( index: 2))
        ->method('sql')
        ->with(
            "SELECT text1 FROM Dummy WHERE `id` = ?",
            ROW_EXPECT_0_1,
            [0=>999])
        ->willReturn(['text1' => 'example text']);
    // Instantiate the InlineEdit object and inject the database mock
    $inlineEdit = new InlineEdit($databaseMock);

    // Act
    $inlineEdit->process($parameters);

    // Assert
    $this->assertEquals( expected: 'ipa', $inlineEdit->getFormName());
    $this->assertEquals( expected: 'text1', $inlineEdit->getColumn());
    $this->assertEquals( expected: '999', $inlineEdit->getRecordId());
    $this->assertEquals( expected: 'text', $inlineEdit->getFeType());
    $this->assertEquals( expected: 'example text', $inlineEdit->getValue());
    $this->assertEquals( expected: 'Dummy', $inlineEdit->getTable());

}
```

Abbildung 44: Unit-Test Screenshot 1

Abbildung 45: Unit-Test Screenshot 2

```php
public function testProcessModeDirect(){

    // Arrange
    $parameters = 'table=Dummy&column=text1&type=text&r=1';
    // Create a mock of the Database class
    $databaseMock = $this->createMock( originalClassName: Database::class);
    // Define the expected behavior of the sql method to return a record
    $databaseMock->expects($this->any())
        ->method('sql')
        ->willReturn(['text1' => 'example text']);
    // Instantiate the InlineEdit object and inject the database mock
    $inlineEdit = new InlineEdit($databaseMock);

    // Act
    $inlineEdit->process($parameters);

    // Assert
    $this->assertEquals( expected: 'Dummy', $inlineEdit->getTable());
    $this->assertEquals( expected: 'text1', $inlineEdit->getColumn());
    $this->assertEquals( expected: 'text', $inlineEdit->getFeType());
    $this->assertEquals( expected: '1', $inlineEdit->getRecordId());
    $this->assertEquals( expected: 'example text', $inlineEdit->getValue());

}
```

Abbildung 43: Unit Test Screenshot 3

```php
public function testCreateFe()
{
    // Arrange
    $recordId = 1;
    $value = 'example text';
    $column = 'text1';
    $table = 'Dummy';
    $feType = 'text';

    // Act
    $formElement = FeFactory::createFe($recordId, $value, $column, $table, $feType, dbIndex: '1');

    // Assert
    $this->assertInstanceOf( expected: FormElementInput::class, $formElement);
}
```

Abbildung 48: Unit Test Screenshot 4

```php
public function testCheckGivenParametersReferenceMissing()
{
    // Arrange
    $parameters = 'form=ipa&r=99';
    $parsedParameters = KeyValueStringParser::explodeKvpSimple($parameters, keyValueDelimiter: '=', listDelimiter: '&');
    $databaseMock = $this->createMock( originalClassName: Database::class);
    $inlineEdit = new InlineEdit($databaseMock);

    // Act
    $result = $inlineEdit->checkGivenParameters($parsedParameters);

    // Assert
    $expectedErrorMessage = 'Missing parameters for special column name "_edit" : fe';
    $expectedErrorCode = ERROR_INLINE_EDIT_MISSING_REQUIRED_PARAMETER;
    $this->assertEquals($expectedErrorMessage, $result['errorMessage']);
    $this->assertEquals($expectedErrorCode, $result['errorCode']);
}
```

Abbildung 46: Unit-Test Screenshot 5

```php
public function testCheckGivenParametersInvalidParameter()
{
    // Arrange
    $parameters = 'tableName=Dummy&columnName=text1';
    $parsedParameters = KeyValueStringParser::explodeKvpSimple($parameters, keyValueDelimiter: '=', listDelimiter: '&');
    $databaseMock = $this->createMock( originalClassName: Database::class);
    $inlineEdit = new InlineEdit($databaseMock);

    // Act
    $result = $inlineEdit->checkGivenParameters($parsedParameters);

    // Assert
    $expectedErrorMessage = 'Invalid parameters for special column name "_edit" provided.';
    $expectedErrorCode = ERROR_INLINE_EDIT_INVALID_PARAMETER_GIVEN;
    $this->assertEquals($expectedErrorMessage, $result['errorMessage']);
    $this->assertEquals($expectedErrorCode, $result['errorCode']);
}
}
```

Abbildung 47: Unit-Test Screenshot 6

```php
public function testRender() {

    // Arrange
    $linkMock = $this->createMock( originalClassName: Link::class);
    $linkMock->expects($this->once())
        ->method('renderLink')
        ->with('p:&r=99&table=Dummy&column=text1&feType=text&value=example text&pageId=123&formName=ipa&formId=999|s|r:8')
        ->willReturn('6479b2b1310ec');
    $databaseMock = $this->createMock( originalClassName: Database::class);
    // Create an instance of the InlineEdit class
    $inlineEdit = new InlineEdit($databaseMock);
    $inlineEdit->setRecordId( recordId: '99');
    $inlineEdit->setTable( table: 'Dummy');
    $inlineEdit->setColumn( column: 'text1');
    $inlineEdit->setFeType( feType: 'text');
    $inlineEdit->setValue( value: 'example text');
    $inlineEdit->setFormName( formName: 'ipa');
    $inlineEdit->setFormId( formId: '999');
    $inlineEdit->setPageId( pageId: '123');
    $inlineEdit->setLink($linkMock);

    // Act
    $actualHtml = $inlineEdit->render();

    // Assert
    $this->assertEquals( expected: '<div class="qfq-inline-edit" data-sip="6479b2b1310ec"><div class="qfq-inline-edit-label">example text</div></div>'
        , $actualHtml);
}
```

Abbildung 50: Unit-Test Screenshot 7

```php
public function testProcessThrowsExceptionNoRecord()
{
    // Arrange
    $parameters = 'table=Dummy&column=text1&type=text&r=99';
    // Create a mock of the Database class
    $databaseMock = $this->createMock( originalClassName: Database::class);
    // Define the expected behavior of the sql method to return an empty result
    $databaseMock->expects($this->any())
        ->method('sql')
        ->willReturn([]);
    // Instantiate the InlineEdit object and inject the database mock
    $inlineEdit = new InlineEdit($databaseMock);

    // Assert
    $this->expectException(UserReportException::class);
    $this->expectExceptionCode( code: ERROR_INLINE_EDIT_RECORD_NONEXISTENT);

    // Act
    $inlineEdit->process($parameters);
}
```

Abbildung 49: Unit-Test Screenshot 8

```php
public function testProcessThrowsExceptionNoFormElement()
{
    // Arrange
    $parameters = 'form=ipa&fe=nonexistentelement&r=1';
    // Create a mock of the Database class
    $databaseMock = $this->createMock( originalClassName: Database::class);
    // Define the expected behavior of the sql method to return a form but empty form element result
    $databaseMock->expects($this->exactly( count: 2))
        ->method('sql')
        ->willReturnOnConsecutiveCalls(
            ['id' => '1', 'tableName' => 'Dummy'],
            []
        );
    // Instantiate the InlineEdit object and inject the database mock
    $inlineEdit = new InlineEdit($databaseMock);

    // Assert
    $this->expectException(UserReportException::class);
    $this->expectExceptionCode( code: ERROR_INLINE_EDIT_FORM_ELEMENT_NONEXISTENT);

    // Act
    $inlineEdit->process($parameters);
}
```

Abbildung 52: Unit-Test Screenshot 9

```php
public function testProcessThrowsExceptionNoForm()
{
    // Arrange
    $parameters = 'form=nonexistentform&fe=text1&r=1';
    // Create a mock of the Database class
    $databaseMock = $this->createMock( originalClassName: Database::class);
    // Define the expected behavior of the sql method to return a result without 'id'
    $databaseMock->expects($this->any())
        ->method('sql')
        ->willReturn([]); // Add the necessary 'tableName' key
    // Instantiate the InlineEdit object and inject the database mock
    $inlineEdit = new InlineEdit($databaseMock);

    // Assert
    $this->expectException(UserReportException::class);
    $this->expectExceptionCode( code: ERROR_INLINE_EDIT_FORM_NONEXISTENT);

    // Act
    $inlineEdit->process($parameters);
}
```

Abbildung 51: Unit-Test Screenshot 10

2.8.4 Integrations-Tests

Generierung einer Textarea nach Klicken auf ein Inline Editing Label	
Durchführung	1. Ich klicke im Frontend auf ein Inline Editing Label.
Erwartetes Ergebnis	Eine Textarea wird generiert. Das Label wird versteckt.
Effektives Ergebnis	Die Textarea wird erfolgreich generiert. Das Label wird korrekt versteckt.
Massnahmen	-

Anzeigen des Labels und Verstecken der Textarea nach Verlassen der Textarea	
Durchführung	1. Ich klicke im Frontend auf ein Inline Editing Label. 2. Die Textarea wird generiert. 3. Dann klicke ich außerhalb der Textarea.
Erwartetes Ergebnis	Das Label wird wieder angezeigt. Die Textarea wird versteckt.
Effektives Ergebnis	Das Label wird erfolgreich wieder angezeigt. Die Textarea wird korrekt versteckt.
Massnahmen	-

Änderung des Werts in der Textarea und Aktualisierung des Labels nach Verlassen der Textarea	
Durchführung	1. Ich klicke auf ein Inline Editing Label im Frontend. 2. Ich ändere den Wert in der generierten Textarea. 3. Dann verlasse ich die Textarea.
Erwartetes Ergebnis	Das Label zeigt den geänderten Wert an.
Effektives Ergebnis	Das Label zeigt erfolgreich den geänderten Wert an.
Massnahmen	-

FormSubmitLog-Eintrag nach Wertänderung	
Durchführung	1. Ich ändere den Wert in der generierten Textarea. 2. Dann verlasse ich die Textarea.
Erwartetes Ergebnis	Es wird ein FormSubmitLog-Eintrag mit allen wichtigen Informationen erstellt.
Effektives Ergebnis	Ein korrekter FormSubmitLog-Eintrag wird erfolgreich erstellt.
Massnahmen	-

Aktualisierung des Datensatzes nach Wertänderung	
Durchführung	1. Ich ändere den Wert in der generierten Textarea. 2. Ich verlasse die Textarea.
Erwartetes Ergebnis	Der entsprechende Datensatz wird mit dem neuen Wert aktualisiert.
Effektives Ergebnis	Der entsprechende Datensatz wird erfolgreich mit dem neuen Wert aktualisiert.
Massnahmen	-

Keine Generierung einer neuen Textarea, wenn bereits eine vorhanden ist	
Durchführung	1. Es gibt bereits eine generierte Textarea. 2. Ich klicke erneut auf das Inline Editing Label.
Erwartetes Ergebnis	Es wird keine neue Textarea generiert. Die bereits vorhandene Textarea wird einfach wieder eingeblendet.
Effektives Ergebnis	Es wird keine neue Textarea generiert. Die bereits vorhandene Textarea wird korrekt wieder eingeblendet.
Massnahmen	-

Kein SQL UPDATE-Statement bei fehlender Wertänderung	
Durchführung	1. Ich ändere den Wert in der generierten Textarea nicht. 2. Ich verlasse die Textarea.
Erwartetes Ergebnis	Es wird kein SQL UPDATE-Statement ausgeführt, da keine Wertänderung stattgefunden hat.
Effektives Ergebnis	Es wird kein SQL UPDATE-Statement ausgeführt, wie erwartet.
Massnahmen	-

Kein FormSubmitLog-Eintrag bei fehlender Wertänderung	
Durchführung	1. Ich ändere den Wert in der generierten Textbox nicht. 2. Ich verlasse die Textbox.
Erwartetes Ergebnis	Es wird kein FormSubmitLog-Eintrag erstellt, da keine Wertänderung stattgefunden hat.
Effektives Ergebnis	Es wird kein FormSubmitLog-Eintrag erstellt, wie erwartet.
Massnahmen	-

Fehlermeldung bei fehlendem Parameter für Special Column Name "_edit" im TYPO3 Backend	
Durchführung	1. Im TYPO3 Backend fehlt der erforderliche Parameter für die Definition der Special Column Name "_edit". 2. Ich klicke auf das Inline Editing Label im Frontend.
Erwartetes Ergebnis	Es wird die entsprechende Fehlermeldung angezeigt.
Effektives Ergebnis	Die richtige Fehlermeldung für den fehlenden Parameter wird erfolgreich angezeigt.
Massnahmen	-

Fehlermeldung bei ungültigem Parameter für Special Column Name "_edit" im TYPO3 Backend	
Durchführung	1. Im TYPO3 Backend wird ein ungültiger Parameter für die Definition der Special Column Name "_edit" verwendet. 2. Ich klicke auf das Inline Editing Label im Frontend.
Erwartetes Ergebnis	Es wird die entsprechende Fehlermeldung für den ungültigen Parameter angezeigt.
Effektives Ergebnis	Die richtige Fehlermeldung für den ungültigen Parameter wird erfolgreich angezeigt.
Massnahmen	-

2.9 Auswerten

2.9.1 Zeitplanung Soll/Ist

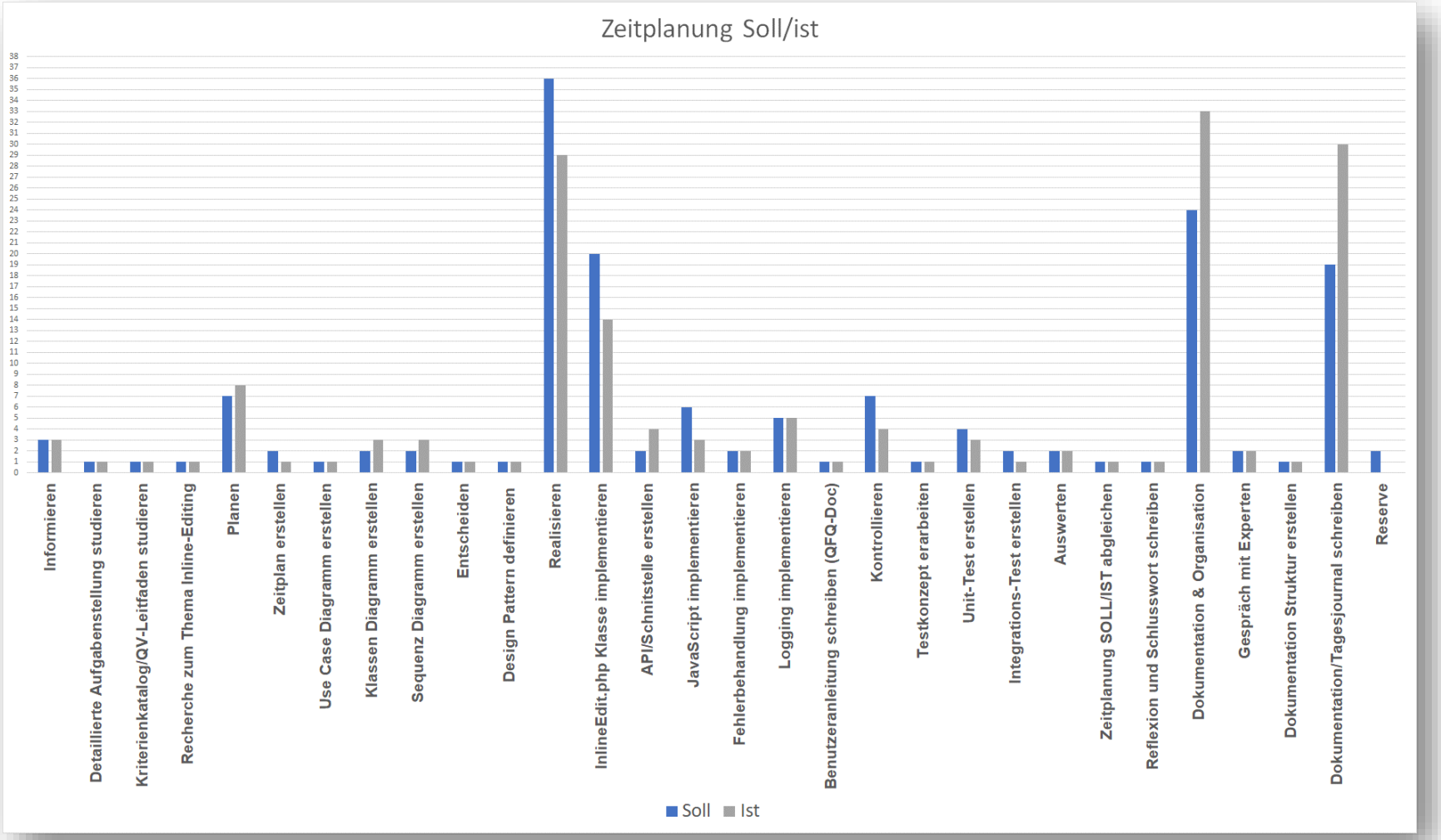


Abbildung 53: Zeitplanung Soll/Ist Abgleich

2.9.2 Ausblick

Da mit meiner IPA nur der Eingabetyp Text realisiert wurde, war von Anfang an klar, dass weitere Eingabetypen implementiert werden können. Die Struktur des von mir geschriebenen Codes hat diese Tatsache berücksichtigt. Je nachdem, welche Aufgaben mir nach dem IPA im Betrieb gegeben werden, habe ich vor diese weiteren Eingabetypen, wie z.B. ein Dropdown für den SQL-Datentyp "Enum" oder einen Datepicker für den SQL-Datentyp "Date", selbst zu implementieren.

2.9.3 Reflexion und Schlusswort

In erster Linie hat mir die Umsetzung, der praktische Teil dieser Projektarbeit sehr viel Spaß gemacht. Vor allem, dass ich die QFQ "Codebase" um ein meiner Meinung nach sehr nützliches Feature erweitern konnte, hat mich motiviert. Die Planung verlief bis auf die Erstellung der Sequenzdiagramme und des Klassendiagramms, welches später noch angepasst werden musste, reibungslos.

Die Entscheidung, mich anfangs schnell in die Realisierungsphase zu stürzen und die Dokumentation etwas zu vernachlässigen, sehe ich einerseits positiv, da ich mich in die praktische Arbeit vertiefen konnte, andererseits musste ich diese Dokumentationsarbeit gegen Ende meiner Arbeit nachholen, was mit der Zeit etwas anstrengend wurde. Es wäre sicher sinnvoll, wenn ich in einer nächsten Arbeit ein gesundes Mittelmaß zwischen Dokumentation und praktischer Arbeit finden könnte.

Wo ich unnötigerweise ein wenig Zeit verloren habe, war das Styling der "Textarea" Außerdem habe ich noch JavaScript-Funktionen geschrieben, die zum Beispiel die "Textarea" und das Label automatisch an die Größe des Textes anpassen, bevor die Seite neu geladen wird. Das war eigentlich nicht die Aufgabe, daher hätte ich die Zeit besser in Punkte investiert, die in die Bewertung einfließen. Zudem habe ich mich für den zeitaufwendigeren Lösungsweg für die Erstellung der "Textarea" entschieden, da diese "on the fly" generiert werden. Die einfachere Lösungsvariante wäre gewesen alle "Textareas" bereits beim Laden der Seite zu generieren und zu verstecken und den entsprechenden Labels zuzuweisen.

Abschliessend bin ich im Großen und Ganzen mit jeder Phase meines Projekts und dem Resultat meiner Projektarbeit zufrieden.

2.10 Glossar

Begriff	Erklärung
Ampersands	Bezeichnung für das Zeichen &
Blur	Art eines Events, der ausgelöst wird, wenn ein Eingabefeld den Fokus verliert.
checkType	Parameter, der in der Form oder FormElement Konfiguration gesetzt werden kann, um die für ein Eingabefeld erlaubten Zeichen zu bestimmen.
encode	Parameter, der in der Form oder FormElement Konfiguration gesetzt werden kann, um die Encodierung zu bestimmen.
Mock	Ein PHPUnit Mock-Objekt ist ein simuliertes Objekt, das das Verhalten eines Teils der Anwendung ausführt, das für den Unit-Test erforderlich ist.
modeSql	Parameter, der in der Form oder FormElement Konfiguration gesetzt werden kann, um die das Element entweder auf show, hidden, required oder readonly zu setzten.
QFQ-Block	Ein QFQ-Content-Element kann aus mehreren QFQ-Blocken bestehen die auch ineinander verschachtelt werden können. Beispiel 10.sql=SELECT id FROM Person
QFQ-Content-Element	Die QFQ-Extension ermöglicht es QFQ-Content-Elemente auf den Seiten zu erstellen.
Sip	Der Schlüssel der nötig ist um die im Sip Store abgelegten Werte abzurufen zukönnen. Beispiel: 64806f22526b7
Sip Store	Speicher, indem Werte Sip codiert, abgelegt werden können.
special column name	Syntax die gebraucht wird, um einer SQL-Abfrage spezielle Funktionalität bei der Generierung des Outputs hinzuzufügen. Beispiel: _edit, _link oder _pdf.
Typo 3 Back End	Hier können Seiten erstellt werden denen zum Beispiel QFQ-Content-Elemente hinzugefügt werden können.

2.11 Verzeichnisse

2.11.1 Quellenverzeichnis

https://refactoring.guru/design-patterns

https://api.jquery.com/

https://mariadb.org/documentation/

2.11.2 Abbildungsverzeichnis

3 Anhang (Teil 3)

3.1 Quellcode

Da es schwierig ist mit angebrachtem Aufwand den ganzen Code in Word mit korrektem Code-Highlighting einzufügen. Darum bitte ich Sie falls nötig im ZIP-Anhang die verschiedenen Klassen und Skripte in einem Code Editor zu öffnen.

3.1.1 InlineEdit.php

```
<?php
/**
 * Created by PhpStorm.
 * User: proess
 * Date: 2/06/23
 * Time: 16:20 PM
 */

namespace IMATHUZH\Qfq\Core\Report;

use CodeException;
use DbException;
use IMATHUZH\Qfq\Core\Database\Database;
use IMATHUZH\Qfq\Core\Helper\KeyValueStringParser;
use IMATHUZH\Qfq\Core\Helper\SqlQuery;
use IMATHUZH\Qfq\Core\Helper\Support;
use IMATHUZH\Qfq\Core\Store\Sip;
use IMATHUZH\Qfq\Core\Store\Store;
use UserFormException;
use UserReportException;

/**
 * Class InlineEdit
 * @package qfq
 */
class InlineEdit {
    /**
     * @var Database
     */
    private Database $database;
    /**
```

```
     * @var Store|null
     */
    private ?Store $store = null;
    /**
     * @var Link
     */
    private Link $link;
    /**
     * @var string|null
     */
    private ?string $table = null;
    /**
     * @var string|null
     */
    private ?string $column = null;
    /**
     * @var string|null
     */
    private ?string $feType = null;
    /**
     * @var string|null
     */
    private ?string $recordId = null;
    /**
     * @var string|null
     */
    private ?string $value = null;
    /**
     * @var string|null
     */
    private ?string $formName = null;
    /**
     * @var string|null
     */
    private ?string $formId = null;
    /**
     * @var string|null
     */
    private ?string $pageId = null;
```

```
/**
 * InlineEdit constructor.
 * Initializes a new instance of the InlineEdit class.
 *
 * @throws UserFormException
 * @throws CodeException
 * @throws UserReportException
 */
public function __construct(Database $database) {
    $this->store = Store::getInstance();
    $this->database = $database;
    $this->link = new Link(new Sip());
}

/**
 * Process the given parameters and map them to the corresponding properties of the InlineEdit object.
 * Example: 'table=Dummy&column=text1&type=text&r=1' or 'form=ipa&fe=text1&r=1'
 *
 * @param string $parameters The parameters to process.
 *
 * @throws UserFormException
 * @throws CodeException
 * @throws DbException
 * @throws UserReportException
 */
public function process(string $parameters) {

    // Convert parameters from string to array
    $parsedParameters = KeyValueStringParser::explodeKvpSimple($parameters,'=','&');

    // Check if 'form', 'fe' and 'r' keys exist
    if (array_key_exists(INLINE_EDIT_FORM, $parsedParameters)
        && array_key_exists(INLINE_EDIT_FORM_ELEMENT, $parsedParameters)
        && array_key_exists(INLINE_EDIT_RECORD_ID, $parsedParameters)) {

        // Set properties from parsedParameters
        $this->formName = $parsedParameters[INLINE_EDIT_FORM];
```

```
        // Get tableName based on formName
        $sql = SqlQuery::selectFormByName($this->formName);
        $form = $this->database->sql($sql[0],ROW_EXPECT_0_1, $sql[1]);

        // Check if sql query found a form
        if(!isset($form[INLINE_EDIT_TABLE_NAME])){
            throw new \UserReportException ('Form "'. $parsedParameters[INLINE_EDIT_FORM] .'" does not exist.'
                , ERROR_INLINE_EDIT_FORM_NONEXISTENT
            );
        }

        // Set properties from parsedParameters
        $this->formId = $form[F_ID];
        $this->table = $form[INLINE_EDIT_TABLE_NAME];
        $this->column = $parsedParameters[INLINE_EDIT_FORM_ELEMENT];
        $this->recordId = $parsedParameters[INLINE_EDIT_RECORD_ID];

        // Get formElement to get its type based on column and formName
        $sql = SqlQuery::selectFormElementByName($this->column, $this->formName);
        $formElement = $this->database->sql($sql[0],ROW_EXPECT_0_1, $sql[1]);

        // Check if sql query found a formElement
        if(!isset($formElement[FE_TYPE])){
            throw new \UserReportException ('FormElement "'
                                            . $parsedParameters[INLINE_EDIT_FORM_ELEMENT]
                                            .'" does not exist on given Form "'
                                            . $this->formName .'".'
                , ERROR_INLINE_EDIT_FORM_ELEMENT_NONEXISTENT
            );
        }

        $this->feType = $formElement[FE_TYPE];

    // Check if 'table', 'column', 'type' and 'r' keys exist
    } elseif (array_key_exists(INLINE_EDIT_TABLE, $parsedParameters)
        && array_key_exists(INLINE_EDIT_COLUMN, $parsedParameters)
        && array_key_exists(FE_TYPE, $parsedParameters)
        && array_key_exists(INLINE_EDIT_RECORD_ID, $parsedParameters)) {
```

```
            // Set properties with the values from the parsed parameters
            $this->table = $parsedParameters[INLINE_EDIT_TABLE];
            $this->column = $parsedParameters[INLINE_EDIT_COLUMN];
            $this->feType = $parsedParameters[FE_TYPE];
            $this->recordId = $parsedParameters[INLINE_EDIT_RECORD_ID];

            // Error handling missing or invalid parameters
        } else {
            $error = $this->checkGivenParameters($parsedParameters);
            throw new \UserReportException ($error[INLINE_EDIT_ERROR_MESSAGE], $error[INLINE_EDIT_ERROR_CODE]);
        }

        // Get value given the recordId, tableName and columnName
        $result = $this->database->sql("SELECT $this->column FROM $this->table WHERE `id` = ?", ROW_EXPECT_0_1, [$this->recordId]);

        // Check if record for given table exists
        if(empty($result)){
            throw new \UserReportException ('Record with id "'. $parsedParameters[INLINE_EDIT_RECORD_ID]
                                            .'" for table "'. $this->table .'" does not exist.',
ERROR_INLINE_EDIT_RECORD_NONEXISTENT);
        }

        // Set value to result of query given the columnName
        $this->value = $result[$this->column];
    }

    /**
     * Create the HTML of the inline edit container and label.
     * Container stores parameters in data-sip attribute.
     *
     * @throws UserFormException
     * @throws CodeException
     * @throws UserReportException
     * @throws DbException
     *
     * @return string HTML of the inline edit container and label.
     */
    public function render(): string {
        // &table has to be named 'table' because of QuickFormQuery.php line 517
```

```
        // Save values in SIP_STORE, so they later can be used in different instance
        $dataSip = $this->link->renderLink('p:&r='. $this->recordId
            .'&table='.$this->table
            .'&column='.$this->column
            .'&feType='.$this->feType
            .'&value='. $this->value
            .'&pageId='. ($this->pageId ?? (strval($this->store->getVar(TYPO3_PAGE_ID, STORE_TYPO3))))
            .'&formName='. ($this->formName ?? 'noForm')
            .'&formId='.strval($this->formId)
            .'|s|r:8'
        );

        // Wrap the value with the HTML tags for the label and the inline edit container, sip is stored in the 'data-sip' attribute of
the container
        $html = Support::wrapTag('<div class="qfq-inline-edit-label">',$this->value);
        $html = Support::wrapTag('<div class="qfq-inline-edit" data-sip="' . $dataSip . '">', $html);

        return $html;
    }

    /**
     * Validates the given parameters for special column name "_edit".
     * This method checks the provided parameters against the expected format for two different modes: "reference" and "direct".
     * In the "reference" mode, the expected parameters are "fe" and "r". In the "direct" mode, the expected parameters are "column",
"type", and "r".
     * If the parameters are missing or invalid, an error message and error code are returned.
     *
     * @param array $parsedParameters An associative array containing the parsed parameters.
     *
     * @return array An associative array containing the error message and error code.
     */
    public function checkGivenParameters(array $parsedParameters): array {
        $missingParameters = array();
        if (isset($parsedParameters[INLINE_EDIT_FORM])) {
            // Version reference mode: 'form=ipa&fe=text1&r=2'
            if (!array_key_exists(INLINE_EDIT_FORM_ELEMENT, $parsedParameters)) {
                $missingParameters[] = INLINE_EDIT_FORM_ELEMENT;
            }
            if (!array_key_exists(INLINE_EDIT_RECORD_ID, $parsedParameters)) {
```

```
                $missingParameters[] = INLINE_EDIT_RECORD_ID;
            }
            $errorMessage = 'Missing parameters for special column name "_edit" : ' . implode(', ', $missingParameters);
            $errorCode = ERROR_INLINE_EDIT_MISSING_REQUIRED_PARAMETER;
        } elseif (isset($parsedParameters[INLINE_EDIT_TABLE])) {
            // Version direct mode: 'table=Dummy&column=text1&type=text&r=2'
            if (!array_key_exists(INLINE_EDIT_COLUMN, $parsedParameters)) {
                $missingParameters[] = INLINE_EDIT_COLUMN;
            }
            if (!array_key_exists(FE_TYPE, $parsedParameters)) {
                $missingParameters[] = FE_TYPE;
            }
            if (!array_key_exists(INLINE_EDIT_RECORD_ID, $parsedParameters)) {
                $missingParameters[] = INLINE_EDIT_RECORD_ID;
            }
            $errorMessage = 'Missing parameters for special column name "_edit": ' . implode(', ', $missingParameters);
            $errorCode = ERROR_INLINE_EDIT_MISSING_REQUIRED_PARAMETER;
        } else {
            $errorMessage = 'Invalid parameters for special column name "_edit" provided.';
            $errorCode = ERROR_INLINE_EDIT_INVALID_PARAMETER_GIVEN;
        }

        return array(INLINE_EDIT_ERROR_MESSAGE => $errorMessage, INLINE_EDIT_ERROR_CODE => $errorCode);
    }

    // Getter functions
    /**
     * Get the form element type.
     *
     * @return string|null The frontend type of the form element, or null if not set.
     */
    public function getFeType(): ?string
    {
        return $this->feType;
    }

    /**
     * Get the record ID associated with the form element.
     *
```

```
     * @return string|null The record ID of the form element, or null if not set.
     */
    public function getRecordId(): ?string
    {
        return $this->recordId;
    }

    /**
     * Get the form name of the form element.
     *
     * @return string|null The form name of the form element, or null if not set.
     */
    public function getFormName(): ?string
    {
        return $this->formName;
    }

    /**
     * Set the form element type .
     *
     * @param string|null $feType The frontend type of the form element.
     *                            Set to null to unset the value.
     *
     * @return void
     */
    public function setFeType(?string $feType): void
    {
        $this->feType = $feType;
    }

    /**
     * Set the record ID associated with the form element.
     *
     * @param string|null $recordId The record ID of the form element.
     *                              Set to null to unset the value.
     *
     * @return void
     */
    public function setRecordId(?string $recordId): void
```

```
    {
        $this->recordId = $recordId;
    }

    /**
     * Set the form name of the form element.
     *
     * @param string|null $formId The form name of the form element.
     *                            Set to null to unset the value.
     *
     * @return void
     */
    public function setFormId(?string $formId): void
    {
        $this->formId = $formId;
    }

}
```

3.1.2 InlineEdit.js

```
var QfqNS = QfqNS || {};

$(document).ready(function() {
    (function(n) {
        $('.qfq-inline-edit').on('click', function() {
            var dataSip = $(this).data('sip');
            var $input = $(this).children('textarea');
            var $label = $(this).children('div.qfq-inline-edit-label');

            if ($input.length === 0) {
                // If the textarea doesn't exist, retrieve it
                retrieveTextbox(dataSip, $label);
            } else {
                $input.removeClass('hidden');
                $input.focus();
                $label.addClass('hidden');
            }
        });
    })(QfqNS);
});

function retrieveTextbox(dataSip, $label) {
    $.ajax({
        url: "typo3conf/ext/qfq/Classes/Api/inlineEditLoad.php?s=" + dataSip,
        method: "POST",
        success: function(response) {
            // Create the textbox and modify its properties
            var $textBox = createTextbox(response);
            $label.addClass('hidden');
            insertAfterLabel($label, $textBox);
            setTextboxHeight($textBox);
            enableAutoGrow($textBox);
            adjustContainerHeight($textBox, $label,'textBox');
            registerBlurHandler($textBox, dataSip, $label);
        },
        error: function() {
            console.error('inlineEditLoad.php API error');
        }
```

```
    });
}

function createTextbox(response) {
    // Create a textbox element from the server response
    var $textBox = $(response);
    $textBox.addClass('qfq-inline-edit-input');
    $textBox.attr('spellcheck', 'false');
    return $textBox;
}

function insertAfterLabel($label, $textBox) {
    // Insert the textbox after the label and set focus on the textbox
    $label.after($textBox);
    $textBox.focus();
}

function setTextboxHeight($textBox) {
    // Set the height of the textbox to match its content
    $textBox.css('height', 'auto');
    $textBox.css('height', $textBox[0].scrollHeight + 'px');
}

function enableAutoGrow($textBox) {
    // Enable the textbox to auto-grow based on its content
    $textBox.on('input', function() {
        this.style.height = 'auto';
        this.style.height = this.scrollHeight + 'px';
    });
}

function adjustContainerHeight($textBox, $label, $mode) {
    // Adjust the height of the container based on the height of the label
    var height;
    if ($mode === 'textBox') {
        height = $textBox.height();
    } else if($mode === 'label'){
        height = $label.height();
        $textBox.height(height);
```

```
    }
    $textBox.parent().parent().height(height);
    $textBox.parent().height(height);
}
function registerBlurHandler($textBox, dataSip, $label) {
    // Register the blur event handler for the textbox
    $textBox.on('blur', function() {
        var updatedValue = $(this).val();

        // Send an AJAX request to update the record in the database
        $.ajax({
            url: "typo3conf/ext/qfq/Classes/Api/inlineEditSave.php?s=" + dataSip,
            method: "POST",
            data: { 'updatedValue': updatedValue },
            success: function(response) {
                $textBox.addClass('hidden');
                $label.text(response).removeClass('hidden');
                adjustContainerHeight($textBox, $label,'label');
            },
            error: function() {
                $textBox.addClass('hidden');
                $label.removeClass('hidden');
            }
        });
    });
```

3.1.3 FeFactory.php

```
<?php
/**
 * Created by PhpStorm.
 * User: proess
 * Date: 2/06/23
 * Time: 16:20 PM
 */
namespace IMATHUZH\Qfq\Core;

use CodeException;
use DbException;
use IMATHUZH\Qfq\Core\Form\AbstractFormElement;
use IMATHUZH\Qfq\Core\Form\FormElementInput;
use IMATHUZH\Qfq\Core\Helper\Support;
use UserFormException;
use UserReportException;
/**
 * Class FeFactory
 * @package qfq
 */
class FeFactory {
    /**
     * Creates and returns an instance of a concrete class that extends the AbstractFormElement class, based on the
given feType.
     *
     * @param string $recordId The record ID.
     * @param mixed $value The value of the form element.
     * @param string $column The column name.
     * @param string $table The table name.
     * @param string $feType The form element type.
     *
     * @throws UserFormException
     * @throws CodeException
     * @throws UserReportException
     * @throws DbException
     *
     * @return AbstractFormElement Instance of concrete FormElement class.
     */
```

```
    public static function createFe(string $recordId, string $value, string $column
        , string $table, string $feType, string $dbIndex):AbstractFormElement {

        // Define needed default values
        $formElement = array(FE_NAME => $column,
            FE_MODE_SQL => '',
            FE_ENCODE => 'specialchar',
            FE_CHECK_TYPE => 'auto',
            FE_MODE => 'show',
            FE_HTML_ID => $column . '-' . $table . '-' . $recordId,
        );

        $htmlFormElementName = $column . '-' . $table . '-' . $recordId;

        // Design Pattern: Factory
        // Instantiate the formElement object based on the feType
        switch ($feType){
            case FE_TYPE_TEXT:
                $formElement[FE_TYPE] = 'text';
                $formElement[FE_SIZE] = '1,0,1';

                #todo would ne nice to not call setFeDefault() for every type, only once would be better
                $formElement = Support::setFeDefaults($formElement);

                $fe = new FormElementInput($formElement, $value, $htmlFormElementName, $dbIndex);
                break;

            case FE_TYPE_SELECT:
                $formElement[FE_TYPE] = FE_TYPE_SELECT;
                break;

            case FE_TYPE_UPLOAD:
                $formElement[FE_TYPE] = FE_TYPE_UPLOAD;
                break;

        }

        return $fe;
```

```
    }
}
```

3.1.4 InlineEditTest.php

```
<?php
/**
 * Created by PhpStorm.
 * User: proess
 * Date: 07/06/23
 * Time: 10:07 PM
 */

namespace IMATHUZH\Qfq\Tests\Unit\Core\Report;

use CodeException;
use DbException;
use IMATHUZH\Qfq\Core\Database\Database;
use IMATHUZH\Qfq\Core\FeFactory;
use IMATHUZH\Qfq\Core\Form\FormElementInput;
use IMATHUZH\Qfq\Core\Helper\KeyValueStringParser;
use IMATHUZH\Qfq\Core\Report\InlineEdit;
use IMATHUZH\Qfq\Core\Report\Link;
use PHPUnit\Framework\TestCase;
use UserFormException;
use UserReportException;

/**
 * Class InlineEditTest
 * @package qfq
 */
class InlineEditTest extends TestCase {

    /**
     * @throws UserFormException
     * @throws CodeException
     * @throws UserReportException
     * @throws DbException
```

```
     */
    public function testProcessModeDirect(){

        // Arrange
        $parameters = 'table=Dummy&column=text1&type=text&r=1';
        // Create a mock of the Database class
        $databaseMock = $this->createMock(Database::class);
        // Define the expected behavior of the sql method to return a record
        $databaseMock->expects($this->any())
            ->method('sql')
            ->willReturn(['text1' => 'example text']);
        // Instantiate the InlineEdit object and inject the database mock
        $inlineEdit = new InlineEdit($databaseMock);

        // Act
        $inlineEdit->process($parameters);

        // Assert
        $this->assertEquals('Dummy', $inlineEdit->getTable());
        $this->assertEquals('text1', $inlineEdit->getColumn());
        $this->assertEquals('text', $inlineEdit->getFeType());
        $this->assertEquals('1', $inlineEdit->getRecordId());
        $this->assertEquals('example text', $inlineEdit->getValue());

    }

    /**
     * @throws UserFormException
     * @throws CodeException
     * @throws UserReportException
     * @throws DbException
     */
    public function testProcessModeReference(){

        // Arrange
        $parameters = 'form=ipa&fe=text1&r=999';
        // Create a mock of the Database class
        $databaseMock = $this->createMock(Database::class);
        // Define the expected behavior of the sql method calls
```

```
        $databaseMock->expects($this->at(0))
            ->method('sql')
            ->with(
                "SELECT  *  FROM `Form` AS f WHERE `f`.`name` LIKE ? AND `f`.`deleted`='no'",
                ROW_EXPECT_0_1,
                ['ipa'],
            )
            ->willReturn(['id' => '999', 'tableName' => 'Dummy']);
        $databaseMock->expects($this->at(1))
            ->method('sql')
            ->with(
                "SELECT fe.* FROM `FormElement` AS `fe` JOIN `Form` AS `f` ON `fe`.`formId` = `f`.`id` WHERE
`fe`.`name` = ? AND `f`.`name` = ?",
                ROW_EXPECT_0_1,
                [0 => 'text1', 1 => 'ipa'])
            ->willReturn(['type' => 'text']);
        $databaseMock->expects($this->at(2))
            ->method('sql')
            ->with(
                "SELECT text1 FROM Dummy WHERE `id` = ?",
                ROW_EXPECT_0_1,
                [0=>999])
            ->willReturn(['text1' => 'example text']);
        // Instantiate the InlineEdit object and inject the database mock
        $inlineEdit = new InlineEdit($databaseMock);

        // Act
        $inlineEdit->process($parameters);

        // Assert
        $this->assertEquals('ipa', $inlineEdit->getFormName());
        $this->assertEquals('text1', $inlineEdit->getColumn());
        $this->assertEquals('999', $inlineEdit->getRecordId());
        $this->assertEquals('text', $inlineEdit->getFeType());
        $this->assertEquals('example text', $inlineEdit->getValue());
        $this->assertEquals('Dummy', $inlineEdit->getTable());

    }
```

```
    /**
     * @throws UserFormException
     * @throws CodeException
     * @throws DbException
     * @throws UserReportException
     */
    public function testProcessThrowsExceptionNoForm()
    {
        // Arrange
        $parameters = 'form=nonexistentform&fe=text1&r=1';
        // Create a mock of the Database class
        $databaseMock = $this->createMock(Database::class);
        // Define the expected behavior of the sql method to return a result without 'id'
        $databaseMock->expects($this->any())
            ->method('sql')
            ->willReturn([]); // Add the necessary 'tableName' key
        // Instantiate the InlineEdit object and inject the database mock
        $inlineEdit = new InlineEdit($databaseMock);

        // Assert
        $this->expectException(UserReportException::class);
        $this->expectExceptionCode(ERROR_INLINE_EDIT_FORM_NONEXISTENT);

        // Act
        $inlineEdit->process($parameters);
    }

    /**
     * @throws UserFormException
     * @throws CodeException
     * @throws UserReportException
     * @throws DbException
     */
    public function testProcessThrowsExceptionNoFormElement()
    {
        // Arrange
        $parameters = 'form=ipa&fe=nonexistentelement&r=1';
        // Create a mock of the Database class
        $databaseMock = $this->createMock(Database::class);
```

```
        // Define the expected behavior of the sql method to return a form but empty form element result
        $databaseMock->expects($this->exactly(2))
            ->method('sql')
            ->willReturnOnConsecutiveCalls(
                ['id' => '1', 'tableName' => 'Dummy'],
                []
            );
        // Instantiate the InlineEdit object and inject the database mock
        $inlineEdit = new InlineEdit($databaseMock);

        // Assert
        $this->expectException(UserReportException::class);
        $this->expectExceptionCode(ERROR_INLINE_EDIT_FORM_ELEMENT_NONEXISTENT);

        // Act
        $inlineEdit->process($parameters);
    }

    /**
     * @throws UserFormException
     * @throws CodeException
     * @throws UserReportException
     * @throws DbException
     */
    public function testProcessThrowsExceptionNoRecord()
    {
        // Arrange
        $parameters = 'table=Dummy&column=text1&type=text&r=99';
        // Create a mock of the Database class
        $databaseMock = $this->createMock(Database::class);
        // Define the expected behavior of the sql method to return an empty result
        $databaseMock->expects($this->any())
            ->method('sql')
            ->willReturn([]);
        // Instantiate the InlineEdit object and inject the database mock
        $inlineEdit = new InlineEdit($databaseMock);

        // Assert
        $this->expectException(UserReportException::class);
```

```
        $this->expectExceptionCode(ERROR_INLINE_EDIT_RECORD_NONEXISTENT);

        // Act
        $inlineEdit->process($parameters);
    }

    /**
     * @throws UserFormException
     * @throws CodeException
     * @throws DbException
     * @throws UserReportException
     */
    public function testRender() {

        // Arrange
        $linkMock = $this->createMock(Link::class);
        $linkMock->expects($this->once())
            ->method('renderLink')
            ->with('p:&r=99&table=Dummy&column=text1&feType=text&value=example
text&pageId=123&formName=ipa&formId=999|s|r:8')
            ->willReturn('6479b2b1310ec');
        $databaseMock = $this->createMock(Database::class);
        // Create an instance of the InlineEdit class
        $inlineEdit = new InlineEdit($databaseMock);
        $inlineEdit->setRecordId('99');
        $inlineEdit->setTable('Dummy');
        $inlineEdit->setColumn('text1');
        $inlineEdit->setFeType('text');
        $inlineEdit->setValue('example text');
        $inlineEdit->setFormName('ipa');
        $inlineEdit->setFormId('999');
        $inlineEdit->setPageId('123');
        $inlineEdit->setLink($linkMock);

        // Act
        $actualHtml = $inlineEdit->render();

        // Assert
```

```
        $this->assertEquals('<div class="qfq-inline-edit" data-sip="6479b2b1310ec"><div class="qfq-inline-edit-
label">example text</div></div>'
            , $actualHtml);
    }

    /**
     * @throws UserFormException
     * @throws CodeException
     * @throws DbException
     * @throws UserReportException
     */
    public function testCreateFe()
    {
        // Arrange
        $recordId = 1;
        $value = 'example text';
        $column = 'text1';
        $table = 'Dummy';
        $feType = 'text';

        // Act
        $formElement = FeFactory::createFe($recordId, $value, $column, $table, $feType,'1');

        // Assert
        $this->assertInstanceOf(FormElementInput::class, $formElement);
    }

    /**
     * @throws UserReportException
     * @throws UserFormException
     * @throws CodeException
     */
    public function testCheckGivenParametersReferenceMissing()
    {
        // Arrange
        $parameters = 'form=ipa&r=99';
        $parsedParameters = KeyValueStringParser::explodeKvpSimple($parameters,'=','&');
        $databaseMock = $this->createMock(Database::class);
        $inlineEdit = new InlineEdit($databaseMock);
```

```
        // Act
        $result = $inlineEdit->checkGivenParameters($parsedParameters);

        // Assert
        $expectedErrorMessage = 'Missing parameters for special column name "_edit" : fe';
        $expectedErrorCode = ERROR_INLINE_EDIT_MISSING_REQUIRED_PARAMETER;
        $this->assertEquals($expectedErrorMessage, $result['errorMessage']);
        $this->assertEquals($expectedErrorCode, $result['errorCode']);
    }

    /**
     * @throws UserReportException
     * @throws UserFormException
     * @throws CodeException
     */
    public function testCheckGivenParametersDirectMissing()
    {
        // Arrange
        $parameters = 'table=Dummy&column=text1';
        $parsedParameters = KeyValueStringParser::explodeKvpSimple($parameters,'=','&');
        $databaseMock = $this->createMock(Database::class);
        $inlineEdit = new InlineEdit($databaseMock);

        // Act
        $result = $inlineEdit->checkGivenParameters($parsedParameters);

        // Assert
        $expectedErrorMessage = 'Missing parameters for special column name "_edit": type, r';
        $expectedErrorCode = ERROR_INLINE_EDIT_MISSING_REQUIRED_PARAMETER;
        $this->assertEquals($expectedErrorMessage, $result['errorMessage']);
        $this->assertEquals($expectedErrorCode, $result['errorCode']);
    }

    /**
     * @throws UserReportException
     * @throws UserFormException
     * @throws CodeException
     */
```

```
    public function testCheckGivenParametersInvalidParameter()
    {
        // Arrange
        $parameters = 'tableName=Dummy&columnName=text1';
        $parsedParameters = KeyValueStringParser::explodeKvpSimple($parameters,'=','&');
        $databaseMock = $this->createMock(Database::class);
        $inlineEdit = new InlineEdit($databaseMock);

        // Act
        $result = $inlineEdit->checkGivenParameters($parsedParameters);

        // Assert
        $expectedErrorMessage = 'Invalid parameters for special column name "_edit" provided.';
        $expectedErrorCode = ERROR_INLINE_EDIT_INVALID_PARAMETER_GIVEN;
        $this->assertEquals($expectedErrorMessage, $result['errorMessage']);
        $this->assertEquals($expectedErrorCode, $result['errorCode']);
    }
}
```

3.1.5 doForm Funktion

```
  /**
     * Process form.
     * $mode=
     *   FORM_LOAD: The whole form will be rendered as HTML Code, including the values of all form elements
     *   FORM_UPDATE: States and values of all form elements will be returned as JSON.
     *   FORM_SAVE: The submitted form will be saved. Return Failure or Success as JSON.
     *   FORM_DELETE:
     *
     * @param string $formMode FORM_LOAD | FORM_UPDATE | FORM_SAVE | FORM_DELETE
     *
     * @return array|string
     * @throws \CodeException
     * @throws \DbException
     * @throws \DownloadException
     * @throws \PhpOffice\PhpSpreadsheet\Exception
     * @throws \PhpOffice\PhpSpreadsheet\Reader\Exception
     * @throws \PhpOffice\PhpSpreadsheet\Writer\Exception
     * @throws \Twig\Error\LoaderError
     * @throws \Twig\Error\RuntimeError
     * @throws \Twig\Error\SyntaxError
     * @throws \UserFormException
     * @throws \UserReportException
     */
    private function doForm($formMode) {
        $data = '';
        $foundInStore = '';
        $flagApiStructureReGroup = true;
        $formModeNew = '';
        $build = null;

        // Check if there is a recordId specified in Bodytext - parse if it is a query
        $rTmp = $this->store->getVar(SIP_RECORD_ID, STORE_TYPO3, SANITIZE_ALLOW_ALL);
        if (false !== $rTmp && !ctype_digit($rTmp)) {
            $rTmp = $this->evaluate->parse($rTmp);
            $this->store->setVar(SIP_RECORD_ID, $rTmp, STORE_TYPO3);
        }
```

```
        $recordId = $this->store->getVar(SIP_RECORD_ID, STORE_TYPO3 . STORE_SIP . STORE_CLIENT . STORE_ZERO,
SANITIZE_ALLOW_DIGIT, $foundInStore);
        $this->setParameterLanguageFieldName();

        $formName = $this->loadFormSpecification($formMode, $recordId, $foundInStore, $formLogMode);
        if ($formName !== false && $formLogMode !== false) {
            return $this->getFormLog($formName, $formLogMode);
        }

        if ($formName === false) {
            switch ($formMode) {
                case FORM_INLINE_EDIT_LOAD:
                    $formModeNew = FORM_INLINE_EDIT_LOAD;
                    break;
                case FORM_INLINE_EDIT_SAVE:
                    $formModeNew = FORM_INLINE_EDIT_SAVE;
                    break;
                case FORM_DELETE:
                    $formModeNew = FORM_DELETE;
                    break;
                case FORM_DRAG_AND_DROP:
                    throw new \CodeException('Missing form in SIP', ERROR_MISSING_FORM);
                default:
                    return '';// No form found: do nothing
            }
        }

        // Check 'session expire' happens quite late, cause it can be configured per form.
        if ($formName !== false) {
            Session::checkSessionExpired($this->formSpec[F_SESSION_TIMEOUT_SECONDS]);
        }

        // Fill STORE_FORM: might need Form.fillStoreVar={{!SELECT ...}}) to provide STORE_VAR - therefore the FORM-
definition should already been processed. #8058
        switch ($formMode) {
            case FORM_UPDATE:
            case FORM_SAVE:
            case FORM_REST:
                $fillStoreForm = new FillStoreForm();
```

```
                $fillStoreForm->process($formMode);

                // STORE_TYPO3 has been filled: fire fillStoreVar again.
                if (!empty($this->formSpec[FE_FILL_STORE_VAR])) {
                    $this->fillStoreVar($this->formSpec[FE_FILL_STORE_VAR]);
                }

                break;
        }

        if ($formName !== false) {
            // Validate (only if there is a 'real' form, not a FORM_DELETE with only a table name).
            // Attention: $formModeNew will be set
            $sipFound = $this->validateForm($foundInStore, $formMode, $formModeNew);

        } else {
            // FORM_DELETE without a form definition: Fake the form with only a tableName.
            $table = $this->store->getVar(SIP_TABLE, STORE_SIP);
            if ($table === false) {
                throw new \UserFormException("No 'form' and no 'table' definition found.", ERROR_MISSING_VALUE);
            }

            $sipFound = true;
            $this->formSpec[F_NAME] = '';
            $this->formSpec[F_TABLE_NAME] = $table;
            $this->formSpec[F_RECORD_LOCK_TIMEOUT_SECONDS] = 1; // just indicate a timeout, the exact timeout is
stored in the dirty record.
            $this->formSpec[F_DIRTY_MODE] = DIRTY_MODE_EXCLUSIVE; // just set a mode,, the exact mode is stored in
the dirty record.
            $this->formSpec[F_PRIMARY_KEY] = F_PRIMARY_KEY_DEFAULT;

            $tmpDbIndexData = $this->store->getVar(PARAM_DB_INDEX_DATA, STORE_SIP);
            if (!empty($tmpDbIndexData)) {
                $this->formSpec[F_DB_INDEX] = $tmpDbIndexData;
                if ($tmpDbIndexData != $this->dbIndexData) {
                    if (!isset($this->dbArray[$tmpDbIndexData])) {
                        $this->dbArray[$tmpDbIndexData] = new Database($tmpDbIndexData);
                    }
                }
```

```
            }
        }

        if (FEATURE_FORM_FILE_SYNC) {
            // FormAsFile: If the record is a Form or FormElement get the new and the old form file name and make
sure both files are writable (before DB changes are made).
            // Note: This can't be done earlier because $formModeNew might be changed in the lines above.
            list($formFileName, $formFileNameDelete) = $this->formAsFileBeforeSave($recordId, $formModeNew);
        }

        // For 'new' record always create a new Browser TAB-uniq (for this current form, nowhere else used) SIP.
        // With such a Browser TAB-uniq SIP, multiple Browser TABs and following repeated NEWs are easily
implemented.
        if ($formMode != FORM_REST) {
            if (!$sipFound || ($formMode == FORM_LOAD && $recordId === 0)) {
                $this->store->createSipAfterFormLoad($formName);
            }
        }

        //Change recordId from Multiform to 0 - No row exception possible
        if (isset($this->formSpec[F_MULTI_MODE]) && $this->formSpec[F_MULTI_MODE] !== 'none' ) {
            $recordId = 0;
            $this->store->setVar(SIP_RECORD_ID, $recordId, STORE_SIP);
        }

        // Fill STORE_BEFORE
        if ($formName !== false && $this->store->getVar($this->formSpec[F_PRIMARY_KEY], STORE_BEFORE) === false) {
            $this->store->fillStoreWithRecord($this->formSpec[F_TABLE_NAME], $recordId,
                $this->dbArray[$this->dbIndexData], $this->formSpec[F_PRIMARY_KEY], STORE_BEFORE);
        }

        // Check (and release) dirtyRecord.
        if ($formModeNew === FORM_DELETE || $formModeNew === FORM_SAVE) {
            $dirty = new Dirty(false, $this->dbIndexData, $this->dbIndexQfq);

            $answer = $dirty->checkDirtyAndRelease($formModeNew, $this->formSpec[F_RECORD_LOCK_TIMEOUT_SECONDS],
                $this->formSpec[F_DIRTY_MODE], $this->formSpec[F_TABLE_NAME], $this->formSpec[F_PRIMARY_KEY],
$recordId, true);
```

```
            // In case of a conflict, return immediately
            if ($answer[API_STATUS] != API_ANSWER_STATUS_SUCCESS) {
                $answer[API_STATUS] = API_ANSWER_STATUS_ERROR;

                return $answer;
            }
        }

        // FORM_LOAD: if there is a foreign exclusive record lock - show form in F_MODE_READONLY mode.
        if ($formModeNew === FORM_LOAD) {
            $dirty = new Dirty(false, $this->dbIndexData, $this->dbIndexQfq);
            $recordDirty = array();
            $rcLockFound = $dirty->getCheckDirty($this->formSpec[F_TABLE_NAME], $recordId, $recordDirty, $msg);
            if (($rcLockFound == LOCK_FOUND_CONFLICT || $rcLockFound == LOCK_FOUND_OWNER) &&
$recordDirty[F_DIRTY_MODE] == DIRTY_MODE_EXCLUSIVE) {
                $this->formSpec[F_MODE_GLOBAL] = F_MODE_READONLY;
            }
        }

        switch ($formModeNew) {
            case FORM_DELETE:
                $build = new Delete($this->dbIndexData);
                break;
            case FORM_REST:
                break;
            case FORM_LOAD:
            case FORM_SAVE:
            case FORM_UPDATE:
            case FORM_DRAG_AND_DROP:

                $tableDefinition = $this->dbArray[$this->dbIndexData]->getTableDefinition($this-
>formSpec[F_TABLE_NAME]);
                $this->store->fillStoreTableDefaultColumnType($tableDefinition);

                // Check if the defined column primary key exist.
                if ($this->store::getVar($this->formSpec[F_PRIMARY_KEY], STORE_TABLE_COLUMN_TYPES) === false) {
                    throw new \UserFormException("Primary Key '" . $this->formSpec[F_PRIMARY_KEY] . "' not found in
table " . $this->formSpec[F_TABLE_NAME], ERROR_INVALID_OR_MISSING_PARAMETER);
                }
```

```
                $this->ifPillIsHiddenSetChildFeToHidden();

                switch ($this->formSpec['render']) {
                    case 'plain':
                        $build = new BuildFormPlain($this->formSpec, $this->feSpecAction, $this->feSpecNative, $this-
>dbArray);
                        break;
                    case 'table':
                        $build = new BuildFormTable($this->formSpec, $this->feSpecAction, $this->feSpecNative, $this-
>dbArray);
                        break;
                    case 'bootstrap':
                        $build = new BuildFormBootstrap($this->formSpec, $this->feSpecAction, $this->feSpecNative,
$this->dbArray);
                        break;
                    default:
                        throw new \CodeException("This statement should never be reached",
ERROR_CODE_SHOULD_NOT_HAPPEN);
                }
                break;

            case FORM_INLINE_EDIT_LOAD:
                $renderer = new FormElementRenderBS3();
                // Create formElement based on variables stored in STORE_SIP
                $fe = FeFactory::createFe($this->store::getVar(INLINE_EDIT_RECORD_ID,STORE_SIP),
                    $this->store::getVar(FE_VALUE,STORE_SIP),
                    $this->store::getVar(INLINE_EDIT_COLUMN,STORE_SIP),
                    $this->store::getVar(INLINE_EDIT_TABLE,STORE_SIP),
                    $this->store::getVar(INLINE_EDIT_FE_TYPE,STORE_SIP),
                    $this->dbIndexData
                );
                // Prepare feAttributes for rendering
                $fe->process();
                // Generate HTML representation of formElement
                $data = $renderer->renderFormElement($fe);
                break;
            case FORM_INLINE_EDIT_SAVE:
                // Get all necessary values for the update
                $table = $this->store->getVar(INLINE_EDIT_TABLE, STORE_SIP);
```

```
                $column = $this->store->getVar(INLINE_EDIT_COLUMN, STORE_SIP);
                $recordId = $this->store->getVar(INLINE_EDIT_RECORD_ID, STORE_SIP);
                $updatedValue = $this->store->getVar(INLINE_EDIT_UPDATED_VALUE, STORE_CLIENT . STORE_ZERO
                    , SANITIZE_ALLOW_ALL);
                $oldValue = $this->store->getVar(FE_VALUE, STORE_SIP, SANITIZE_ALLOW_ALL);
                // Check if changes were made
                if($updatedValue !== $oldValue){
                    // Update the record with the new value
                    $this->dbArray[$this->dbIndexQfq]->sql("UPDATE $table SET $column = ? WHERE `id` = ?"
                        ,ROW_EXPECT_1,[$updatedValue,$recordId]);
                    $data = $updatedValue;
                    // Prepare formSpec for logging
                    $this->formSpec[F_DO_NOT_LOG_COLUMN] = '';
                    $this->formSpec[F_ID] = $this->store->getVar(FE_FORM_ID, STORE_SIP) ?? '0';
                    $this->formSpec[F_NAME] = $this->store->getVar(INLINE_EDIT_FORM_NAME, STORE_SIP);
                    // Log to formSubmitLog table
                    $this->logFormSubmitRequest(FORM_INLINE_EDIT_SAVE);
                } else {
                    $data = $oldValue;
                }
                break;

            default:
                throw new \CodeException("This statement should never be reached", ERROR_CODE_SHOULD_NOT_HAPPEN);
        }

        $formAction = new FormAction($this->formSpec, $this->dbArray, $this->phpUnit);
        switch ($formModeNew) {
            case FORM_LOAD:
                $formAction->elements($recordId, $this->feSpecAction, FE_TYPE_BEFORE_LOAD);

                // Build FORM
                $data = $build->process($formModeNew);


                $tmpClass = is_numeric($this->formSpec[F_BS_COLUMNS]) ? ('col-md-' . $this->formSpec[F_BS_COLUMNS]) :
$this->formSpec[F_BS_COLUMNS];
//                $data = Support::wrapTag("<div class='" . 'col-md-' . $this->formSpec[F_BS_COLUMNS] . "'>", $data);
                $data = Support::wrapTag('<div class="' . $tmpClass . '">', $data);
```

```
            $data = Support::wrapTag('<div class="row">', $data);
            $formAction->elements($recordId, $this->feSpecAction, FE_TYPE_AFTER_LOAD);
            break;

        case FORM_UPDATE:
            $formAction->elements($recordId, $this->feSpecAction, FE_TYPE_BEFORE_LOAD);
            // data['form-update']=....
            $data = $build->process($formModeNew);
            $formAction->elements($recordId, $this->feSpecAction, FE_TYPE_AFTER_LOAD);
            break;

        case FORM_DELETE:
            $formAction->elements($recordId, $this->feSpecAction, FE_TYPE_BEFORE_DELETE);

            $build->process($this->formSpec[F_TABLE_NAME], $recordId, $this->formSpec[F_PRIMARY_KEY]);

            $formAction->elements($recordId, $this->feSpecAction, FE_TYPE_AFTER_DELETE);
            break;

        case FORM_SAVE:
            $this->logFormSubmitRequest(FORM_SAVE);

            $recordId = $this->store->getVar(SIP_RECORD_ID, STORE_SIP . STORE_TYPO3);

            // SAVE
            $save = new Save($this->formSpec, $this->feSpecAction, $this->feSpecNative, $this->feSpecNativeRaw);

            $rc = $save->process($recordId);

            if ($formMode == FORM_REST) {
                $data = $this->doRestPostPut($rc);
                $flagApiStructureReGroup = false;
                break;
            }

            $this->setForwardModePage();

            // Logic: If a) r=0 and
```

```
                //           b) final: (forwardMode=='auto' and User presses only 'save' (not 'save & close')) OR
(forwardMode=='no')
                // then the client should reload the current page with the newly created record. A new SIP is
necessary!
                $getJson = true;
                if (0 == $this->store->getVar(SIP_RECORD_ID, STORE_SIP)
                    && (($this->formSpec[F_FORWARD_MODE] == F_FORWARD_MODE_AUTO
                            && API_SUBMIT_REASON_SAVE == $this->store->getVar(API_SUBMIT_REASON, STORE_CLIENT .
STORE_EMPTY, SANITIZE_ALLOW_ALNUMX)
                        ) || $this->formSpec[F_FORWARD_MODE] == F_FORWARD_MODE_NO)) {
                    $this->formSpec = $this->buildNSetReloadUrl($this->formSpec, $rc);
                    $getJson = false;
                }

                if ($getJson) {

                    // Values of FormElements might be changed during 'afterSave': rebuild the form to load the new
values. Especially for non primary template groups.
                    $feSpecNative = $this->getNativeFormElements(SQL_FORM_ELEMENT_NATIVE_TG_COUNT, [$this-
>formSpec[F_ID]], $this->formSpec);
                    $parameterLanguageFieldName = $this->store->getVar(SYSTEM_PARAMETER_LANGUAGE_FIELD_NAME,
STORE_SYSTEM);
                    $feSpecNative = HelperFormElement::setLanguage($feSpecNative, $parameterLanguageFieldName);

                    $this->feSpecNative = HelperFormElement::setFeContainerFormElementId($feSpecNative, $this-
>formSpec[F_ID], $recordId);

                    $data = $build->process($formModeNew, false, $this->feSpecNative);
                }
                break;

            case FORM_DRAG_AND_DROP:
                $formAction->elements($recordId, $this->feSpecAction, FE_TYPE_BEFORE_LOAD);

                $dragAndDrop = new DragAndDrop($this->formSpec);
                $data = $dragAndDrop->process();
                $flagApiStructureReGroup = false;

                $formAction->elements($recordId, $this->feSpecAction, FE_TYPE_AFTER_LOAD);
```

```
                break;

            case FORM_REST:
                $flagApiStructureReGroup = false;
                $data = $this->doRestGet();
                break;
            case FORM_INLINE_EDIT_LOAD:
                // Do nothing
                break;
            case FORM_INLINE_EDIT_SAVE:
                // Do nothing
                break;
            default:
                throw new \CodeException("This statement should never be reached", ERROR_CODE_SHOULD_NOT_HAPPEN);
        }

        if ($flagApiStructureReGroup && is_array($data)) {
            // $data['element-update']=...
            $data = $this->groupElementUpdateEntries($data);
        }

        if (FEATURE_FORM_FILE_SYNC) {
            // export Form to file, if loaded record is a Form/FormElement
            $this->formAsFileAfterSave($formFileName, $formModeNew, $formFileNameDelete);
        }

        $unitTestRender = $this->store::getVar(SYSTEM_UNIT_TEST_FORM_CONTENT, STORE_SYSTEM);
        if (isset($unitTestRender) && $unitTestRender !== false) {
            $data = $unitTestRender;
        }

        return $data;
    }
```

3.1.6 CSS

```
.qfq-inline-edit-input {
  min-height:16px!important;
  padding:0 0!important;
  margin-top: -1px!important;
  margin-left: -1px!important;
  position: absolute!important;
  z-index:9999!important;//needed so textarea is always in front
  resize: both;
  line-height: inherit!important;
  font-size: 13px!important;
  overflow: hidden!important;
  width: 101%;//needed for label textarea matching width
}

.qfq-inline-edit {
  min-height: 20.8px;
  position:relative;
  top: 0;
}

.qfq-inline-edit-label {
  white-space: pre-wrap;
}
```

3.1.7 PHP Code Guidelines

- Vor der Definition einer neuen Klasse sind Autor und Erstellungsdatum anzugeben.
- Variablen folgen der camelCase Schreibweise.
- Konstanten sind in Grossbuchstaben zu schreiben.
- Beinhaltet der Name einer Konstante
- zwei oder mehr Wörter, dann sind diese mit einem Unterstrich (_) zu trennen.
- Funktions- und Methoden-Namen sind in camelCase zu schreiben.
- Oberhalb des Kopfs jeder Funktion/Methode ist eine Beschreibung in einem mehrzeiligen
- Kommentar zu erstellen.
- Darin sollen Zweck, Parameter und Rückgabewerte (inkl.
- Datentyp) beschrieben sein.
- Verschachtelter Code muss mit genau einem Tab pro Verschachtelungs-Ebene eingerückt
- werden.
- Es sollen keine Leerzeichen zum Einrücken verwendet werden.
- Kommentare dürfen verwendet werden, wenn diese zum besseren Verständnis des Codes
- führen.
- Einzeilige Kommentare sollen mit // eingeleitet werden. Mehrzeilige Kommentare
- sollen mit /** eingeleitet und mit */ abgeschlossen werden.

```php
<?php
/**
 * Created by PhpStorm.
 * User: enured
 * Date: 5/13/22
 * Time: 09:00 AM
 */

namespace Classes\Core\Helper;

const CONSTANT_ONE = 'muster1';
const CONSTANT_TWO = 'muster2';

/**
 * Class ExampleClass
 * @package qfq
 */
class ExampleClass {

    /**
     * Description of the following function is here.
     *
     * detailed description fo the following function.
     *
     * @param string $givenString
     * @param string $givenStringTwo
     *
     * @return string
     */
    public static function addString(string $givenString, string $givenStringTwo = 'defaultExample') {

        // Commentary here if needed
        $totalString = $givenString.$givenStringTwo;

        return $totalString;
    }
}
```