Interaktives Tutorial System für eine Webanwendung

MASTERARBEIT

Markus Goller

Eingereicht am 15. September 2022

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

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

Friedrich-Alexander-Universität
Technische Fakultät

Eidesstattliche Erklärung

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

Erlangen, 15. September 2022

Lizenz

Erlangen, 15. September 2022

Abstract

Web applications can have a high level of functionality and complexity, making them difficult for users to use intuitively. Users can then resort to instructions or tutorials in order to learn how to use the application correctly. Tutorials that not only impart knowledge but also explain the application to the user while interacting with it are called interactive tutorials.

QDAcity is a cloud-based web application that allows users to collaborate on qualitative data analysis. The functionality provided for qualitative data analysis and the different areas of QDAcity are not easy to understand, especially for new users, and confront them with challenges.

In this thesis, an interactive tutorial system for QDAcity is presented that explains the functionality and areas of the application to users through different tutorials, allows users to create their own tutorials, and lets users share their own tutorials with other users.

Zusammenfassung

Webanwendungen können über ein hohes Maß an Funktionalität und Komplexität verfügen und dadurch für Nutzer nur schwer intuitiv nutzbar sein. Nutzer können dann auf Anleitungen oder Tutorials zurückgreifen und so erlernen wie die Anwendung richtig verwendet werden kann. Tutorials, die nicht nur Wissen vermitteln, sondern dem Nutzer während der Interaktion mit der Anwendung diese erklären, werden als interaktive Tutorials bezeichnet.

QDAcity ist eine cloudbasierte Webanwendung, die Nutzern die Möglichkeit bietet gemeinsam qualitative Datenanalyse zu betreiben. Die für qualitative Datenanalyse bereitgestellte Funktionalität und verschiedenen Bereiche von QDAcity sind gerade für neue Nutzer nicht leicht zu verstehen und stellen diese vor Herausforderungen.

In dieser Arbeit wird ein interaktives Tutorial System für QDAcity vorstellt, das Nutzern die Funktionalität und die Bereiche der Anwendung durch verschiedene Tutorials erklärt, das Erstellen von eigenen Tutorials ermöglicht und von Anwendern erstellte Tutorials mit anderen Nutzern teilen lässt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

Abkürzungsverzeichnis

API Application Programming Interface

ATT Attraktivität

CAQDAS Computerunterstützte qualitative Datenanalyse Software

CI/CD Continuous Integration/Continuous Delivery

CRUD Create Read Update Delete

CSS Cascading Style Sheet

DOM Document Object Model

E2E End-to-End

FA Funktionale Anforderungen

GAE Google App Engine

GCP Google Cloud Plattform

HTTP Hypertext Transfer Protocol

HQ Hedonische Qualität

i18n Internationalisierung

ICA Intercoder Agreement

JDO Java Data Objects

JS JavaScript

LOC Lines of Code

NFA Nicht-Funktionale Anforderungen

NYT Nailing Your Thesis

PDF Portable Document Format

PQ Pragmatische Qualität

QDA Qualitative Datenanalyse

REST Representational State Transfer

RTF Rich Text Format

RPC Remote Procedure Call

UEQ User Experience Questionnaire

UI User Interface

UML Unified Modeling Language

UX User Experience

1 Einleitung

Mit dem Internet hat sich die Art und Weise, wie Wissen erlangt und vermittelt werden kann, grundlegend verändert. Das Internet bietet freien Zugang zu einem nahezu unbegrenzten Angebot an Wissen. Es ist im Gegensatz zu früheren Zeiten nicht mehr nötig, Bücher auszuleihen oder zu kaufen und es finden sich zu so gut wie jedem Themengebiet Informationen online. (McNeely & Wolverton, 2008).

Doch nicht nur in Textform, sondern besonders auch in Gestalt von Videos kann nun Wissen vermittelt werden. Die größte Plattform dafür ist YouTube mit 2.56 Mrd. Nutzern monatlich[1] und einem Upload von 400 Stunden Videomaterial pro Minute[2]. Ein Drittel dieser Nutzer schaut sich dabei lieber wöchentlich ein Tutorial oder Anleitungsvideo an[3], um ein Produkt kennenzulernen, als sich die Anleitung eines Produktes durchzulesen[4].

Neben Tutorials, die allgemeines Wissen vermitteln oder bei der Problemlösung helfen, gibt es auch Tutorials für Software und Webanwendungen. Webanwendungen können neben Text- und Videoanleitungen auch auf Tutorials setzen, die in die Anwendung integriert sind. Oft wird dies in Form von Tipps oder *Ersten Schritten* innerhalb der Anwendung umgesetzt, um den Nutzer interaktiv in die Funktionalität einzuführen. Dem Nutzer werden aktiv die einzelnen Funktionen erklärt und die verschiedenen Bereiche der Anwendung anschaulich und praktisch gezeigt. Dies wird immer wichtiger, da Webanwendungen immer komplexer werden, mehr Funktionalität beinhalten und Nutzer zwischen verschieden Bereichen und Diensten der Anwendung navigieren können müssen (Abrahao et al., 2002).

In dieser Arbeit soll ein interaktives Tutorial System für QDAcity erstellt werden, einer cloudbasierten Webanwendung für qualitative Datenanalyse. Das System soll Tutorials für die verschiedenen Bereiche der Webanwendung bereitstellen und Nutzern die Möglichkeit geben, Tutorials selbst zu erstellen und miteinander zu teilen.

[1] https://de.statista.com/themen/162/youtube/#dossierKeyfigures
[2] https://www.brandwatch.com/de/blog/statistiken-youtube/
[3] https://www.meltwater.com/de/blog/youtube-statistiken
[4] https://bit.ly/3d1wgT8

2 QDAcity

QDAcity ist eine computerunterstützte qualitative Datenanalyse Software (CAQDAS) in Form einer Webanwendung. Methoden zur Qualitative Datenanalyse (QDA) konzentrieren sich auf das Erheben der relevanten Informationen aus der unstrukturierten Menge an qualitativen Daten, dem Interpretieren dieser Daten und dem Abstrahieren von diesen (Kaufmann, 2021). Im Folgenden wird beschrieben mit welchen Funktionen QDAcity QDA unterstützt und wie der technische Aufbau von QDAcity ist.

2.1 Funktionalität

QDAcity verfügt über eine Vielzahl von Funktionen, die den Nutzer bei QDA unterstützen sollen. Die Funktionen können in drei Bereiche aufgeteilt werden. So gibt es den Projektbereich, den Coding Editor und den Kursbereich.

2.1.1 Projektbereich

Der Projektbereich stellt die Übersicht eines Projektes dar. Innerhalb von Projekten können Nutzer zusammenarbeiten und in den Coding Editor des Projektes gelangen, der den zentralen Teil für die QDA übernimmt. Um in den Projektbereich zu gelangen, muss zunächst ein Projekt erstellt werden. Die Projektübersicht besteht aus den folgenden Teilen:

1. Verwaltung der Nutzer im Projekt
2. Projektbeschreibung
3. Projektstatistiken
4. Intercoder Agreement
5. To-Do's
6. Revisionshistorie
7. Projekteinstellungen

Verwaltung der Nutzer im Projekt QDAcity unterstützt das Zusammenarbeiten von mehreren Nutzern innerhalb eines Projektes. Nutzer können so auch parallel an einem Dokument arbeiten. Dafür müssen diese zuerst in das Projekt eingeladen werden. Nutzern kann der Zugriff auf ein Projekt auch wieder entzogen werden.

Projektbeschreibung Die Beschreibung des Projektes wird in der Projektübersicht angezeigt und kann dort auch geändert werden.

Projektstatistiken Die Statistiken zu einem Projekt werden in der Projektübersicht dargestellt. Dort werden die Anzahl der analysierten Dokumente, die Anzahl der erstellten Codes, die Anzahl wie oft diese insgesamt verwendet wurden und die Saturation dargestellt.

Intercoder Agreement Das Intercoder Agreement (ICA) stellt dar, inwiefern sich die Analyse von Nutzern innerhalb eines Projektes unterscheidet. Dabei können Unterschiede zwischen den Codings in ausgewählten Dokumenten durch statistische Methoden wie F-Measure, Krippendorf's Alpha und Fleiss' Kappa ermittelt werden.

To-Do's In einem Projekt können To-Do's erstellt, geändert und gelöscht werden. Den To-Do's können Nutzer und Aufgaben zugeordnet werden und die To-Do's gelten als vollendet, wenn alle Aufgaben erledigt wurden.

Revisionshistorie Für ein Projekt können Revisionen erstellt werden, wobei eine Revision einen Snapshot des Projektes zum Zeitpunkt der Erstellung darstellt. In der Revisionshistorie werden alle Revisionen, startend mit Revision 0, dargestellt.

Einstellungen In der Projektübersicht können auch die Einstellungen eines Projektes geändert werden, wie in Abbildung 2.1 zu sehen ist. Dabei können zum einen verschiedene Editoren für den Coding Editor des Projektes aktiviert oder deaktiviert werden. Zudem kann eingestellt werden, wie stark sich die verschiedenen Änderungsarten auf die Saturation auswirken.

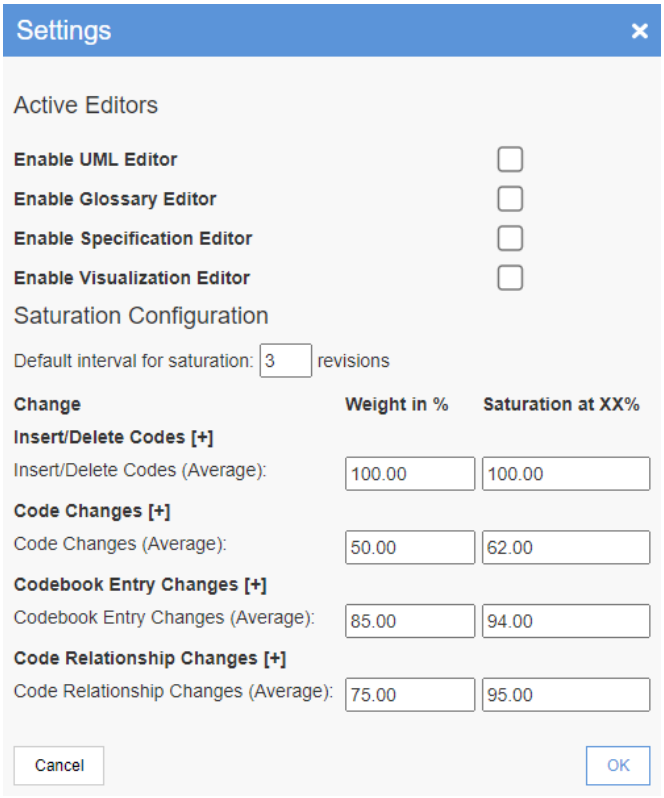

Abbildung 2.1: Einstellungen in einem Projekt in QDAcity.

2.1.2 Coding Editor

Über die Projektübersicht kann der Coding Editor erreicht werden, der aus der Editorauswahl, einer Dokumentenübersicht, dem Codesystem und dem ausgewählten Editor besteht.

Editorauswahl

Je nach Einstellung in der Projektübersicht und ausgewähltem Dokument, kann zwischen den Editor Modi *Coding*, *Editor*, *UML*, *Glossary*, *Specification* und *Visualisierung* gewählt werden. Besonders wichtig sind die folgenden beiden Modi:

- *Coding*: Im Coding Modus des Editor können die Dokumente nur gelesen und Textstellen markiert werden, um Codes anzuwenden.
- *Editor*: Der Editor Modus stellt einen Texteditor dar und ist nur für selbst erstellte Dokumente verfügbar. Mit diesem können diese Dokumente verändert werden.

Dokumentenübersicht

Alle Dokumente des Projektes werden in der Dokumentenübersicht angezeigt. Der Nutzer kann dort verschiedene Arten von Dokumenten wie PDF-, RTF- oder Audiodatein hochladen, um z.B. Interviews zu analysieren. Dokumente können zudem umbenannt oder gelöscht werden. In der Übersicht können Dokumente aufsteigend, absteigend oder nach Belieben sortiert werden.

Codesystem

Für die Analyse können in einem hierarchischen Codesystem Codes hinzugefügt, umbenannt, verschoben, favorisiert oder gelöscht werden. Mit Hilfe dieser Codes können Codings erstellt werden. Codings sind Zuweisungen von Codes zu bestimmten Textpassagen in einem Dokument. Die Eigenschaften von Codes können hier bearbeitet werden und diesen Informationen hinzugefügt werden, wie z.B. in welchen Situationen diese zu verwenden sind und in welchen nicht. Des Weiteren können eingehende und ausgehende Beziehungen zwischen Codes in einem Metamodell definiert werden.

2.1.3 Kursbereich

Im Kursbereich können Kurse mit untergeordneten Semesterkursen erstellt werden. In den Semesterkursen können Aufgaben erstellt werden, die von den Teilnehmern bearbeitet werden sollen. Basierend auf existierenden Projekten und Revisionen können verschiedene Arten von Aufgaben definiert werden, in denen die Teilnehmer qualitative Datenanalyse betreiben können. Im Anschluss kann ermittelt werden, wie stark die Analyse der Teilnehmer der Analyse des Projektes beziehungsweise der Revision ähnelt, die zum Erstellen der Aufgabe genutzt wurde. Dieser Bereich wird bereits in der Vorlesung Nailing Your Thesis (NYT) des Open Source Departements genutzt. Dort nehmen Studenten des Kurses an dem Semesterkurs teil und müssen im Laufe des Semesters verschiedene Aufgaben bearbeiten.

2.2 Technische Übersicht

QDAcity ist eine cloudbasierte Webanwendung mit einer Client-Server Architektur, die in Abbildung 2.2 dargestellt wird, bestehend aus einem JavaScript Client im Frontend und einem Java RESTful Webservices im Backend, die über HTTP Requests miteinander kommunizieren.

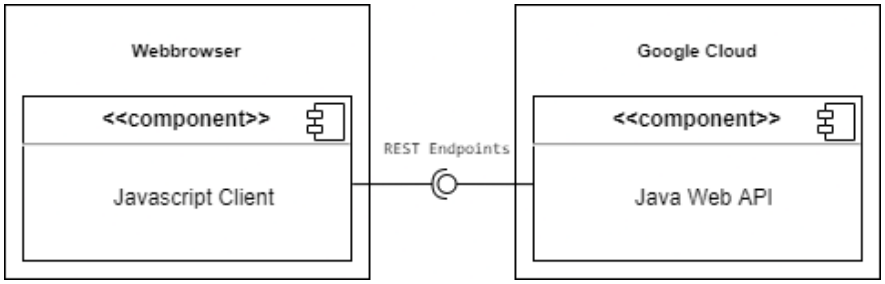

Abbildung 2.2: Komponenten Architektur von QDAcity.

2.2.1 Frontend

Das Frontend besteht aus einem JavaScript Client, der mit Hilfe der JavaScript Bibliothek React[1] implementiert wird. React ermöglicht es dabei die komplexe Benutzeroberfläche in vielen wiederverwendbaren React Komponenten zu implementieren, den Zustand des Clients und der Daten zu verwalten und bietet unter anderem auch Lösungen für Internationalisierung an.

2.2.2 Backend

Im Backend handelt es sich um eine Java 8 Anwendung, die eine RESTful API darstellt und REST Endpunkte für das Erstellen, Lesen, Updaten und Löschen (CRUD) von Daten bereitgestellt. Dafür wird das API-Verwaltungssystem Cloud Endpoints[2] von Google verwendet.

Google Cloud Endpoints sind ein Teil der Google Cloud Plattform (GCP) und einer der GCP Services, die von QDAcity genutzt werden. So wird QDAcity unter anderem über die Google App Engine (GAE)[3] gehostet, die es ermöglicht, Anwendungen in der Google Cloud zu betreiben.

Zur Speicherung der Daten im Backend wird GAE und die NoSQL Datenbank Firestore im Datastore Modus[4] verwendet. Um auf die Daten zuzugreifen, wird

[1] https://reactjs.org/
[2] https://cloud.google.com/endpoints/docs
[3] https://cloud.google.com/appengine?hl=de#section-4
[4] https://cloud.google.com/appengine/docs/legacy/standard/java/using-cloud-datastore

die Java Data Objects (JDO) Schnittstelle der Google App Engine SDK[5] genutzt, die auf dem Open Source Plugin DataNucleus basiert.

[5]https://cloud.google.com/appengine/docs/standard/java/datastore/jdo/overview-dn2

3 Tutorials

Tutorials dienen dazu, dem Nutzer des Tutorials Wissen zu vermitteln. Dabei kann neben der Art des Tutorials zunächst unterschieden werden, in welchem Szenario das Tutorial eingesetzt wird. So kann entweder der Umgang mit einer (Web)-Anwendung oder einem anderen Programm oder explizites Wissen über ein spezifisches Thema vermittelt werden. So wurde untersucht wie Bachelorstudenten mit Hilfe von Tutorials Quantenmechanik besser verstehen können (Singh, 2008) und evaluiert wie Studenten ihr Wissen über den zentralen Grenzwertsatz mit Hilfe von Tutorials verbessern können (Aberson et al., 2000). Dabei wurde der Effekt auf die Studenten auch durch Tests ermittelt, die vor und nach den Tutorials durchgeführt wurden und eine Verbesserung der Leistungen durch die Tutorials darlegten.

In Kapitel 3.1 werden zunächst die verschiedenen Arten und Ausprägungen von Tutorials vorgestellt und in Kapitel 3.2 wird eine geeignete Tutorialart für QDAcity ausgewählt.

3.1 Tutorialvarianten

Tutorials unterscheiden sich in vielen Aspekten, wie dem Zweck oder der Form des Tutorials. In diesem Kapitel werden verschieden Varianten von Tutorials mit ihren Vor- und Nachteilen vorgestellt.

Softwareanwendungen beinhalten oft zu viel Funktionalität, um vom Nutzer intuitiv verstanden werden zu können. Deshalb werden Tutorials angeboten, die dem Nutzer helfen sollen die Anwendung besser zu verstehen. Die Analyse der Verwendung von Tutorials, in Text- und Videoform, bei neuen Softwareanwendungen hat dabei gezeigt, dass weder Text- noch Videotutorials zu favorisieren sind, sondern die verschiedenen Arten Vor- und Nachteile haben (Käfer et al., 2017). Videotutorials erwiesen sich als motivierender auf die Nutzer und wurden daher zunächst bevorzugt für den Einstieg in die Anwendung gewählt, sind jedoch deutlich aufwändiger als Texttutorials. Die Nutzer griffen im Verlauf häufig auf Texttutorials zurück, um gezielt bestimmte Aspekte nachzulesen.

Als besondere Variante der Videotutorials sind sogenannte Screencast Tutorials zu nennen, bei denen der Bildschirm während der Anwendung aufgezeichnet wird und dies zudem kommentiert werden kann (Kawaf, 2019). Durch die Screencast Tutorials werden dem Nutzer direkt gezeigt, wo sich Funktionen innerhalb der Anwendung befinden und wie diese genutzt werden können. Am Beispiel von Studenten, die Statistik durch Tutorials lernen, konnte gezeigt werden, dass Screencast Tutorials deutliche Vorteile gegenüber Texttutorials haben (Lloyd & Robertson, 2012). Dieser Vorteil kann auf die Theorie des Multimedia Lernens zurückgeführt werden, wodurch ein besseres Lernergebnis erreicht wird, wenn Wörter und Bilder kombiniert werden, anstatt diese einzeln zu nutzen (Mayer, 2009).

Neben der Unterscheidung von Text-, Video und Screencast Tutorials, können diese auch in passive und interaktive Tutorials eingeordnet werden und ob diese in Softwareanwendungen integriert oder extern sind. Die Eigenschaften, Unterschiede, Vor- und Nachteile dieser verschiedenen Tutorialarten werden in Tabelle 3.1 dargestellt (Thelen et al., 2013). So handelt es sich bei Text-, Video- und Screencast Tutorials um passive und externe Tutorials, wobei Screencast Tutorials auch interaktive Komponenten beinhalten können. Passive Tutorials vermitteln nur Informationen und reagieren dabei nicht auf Eingaben des Nutzers.

Interne Tutorials sind direkt in die Anwendung eingebunden, können unmittelbar auf Nutzeraktionen reagieren und diese auch validieren. Eine der häufigsten Varianten von internen interaktiven Tutorials stellt die Funktion *Erste Schritte* von Softwareanwendungen dar, die dem Anwender bei der Nutzung helfen und die grundlegenden Funktionen vermitteln soll. Ein Vorteil der internen Tutorials ist, dass der Anwender nicht den Kontext wechseln und von einem Dokument oder Video zur Softwareanwendung springen muss, sondern sich direkt in der Anwendung befindet und dort die einzelnen Funktionen kennenlernen und ausprobieren kann. Dieser Vorteil kommt jedoch mit dem Nachteil einher, dass interne Tutorials direkt mit der Entwicklung der Softwareanwendung verbunden sind. Interne Tutorials sind somit aufwändiger als externe und die Entwicklung und Wartung der Tutorials kann sich als undurchsichtig und komplexer erweisen (Thelen et al., 2013).

Um die Nachteile von integrierten und externen Tutorials zu vermeiden, ohne die Vorteile zu verlieren, wurde das Framework Tutonium entwickelt, das auf den HTML-Baum der Webanwendung mittels DOM zugreift und diesen so manipulieren kann, dass die Tutorialinhalte direkt im Browser angezeigt werden. Das Framework liefert zudem eine Autorenumgebung, über die Tutorials mit Hilfe des Testframeworks Selenium erstellt werden können (Thelen et al., 2013). Obwohl durch das Framework die Tutorials nicht direkt in der Webanwendung verankert werden mussten, geht die Verwendung mit einigen Einschränkungen und Probleme einher. So muss durch die *Same-Origin-Policy* der für das Tutorial

erzeugte Code von dem gleichen Server kommen wie die Anwendung. Auch müssen die Ersteller der Tutorials Kenntnisse über das Testframework Selenium und über Webtechnologien wie CSS und DOM haben. Problematisch ist vor allem auch die Abhängigkeit der Tutorials von dem ausgelieferten HTML-Baum. Für funktionsfähige Tutorials müssen eindeutige Selektoren auf Elemente innerhalb des HTML-Baums vorhanden sein, da es sonst zu falsch ausgewählten Elementen kommen kann und bei Änderungen der Benutzeroberfläche der Anwendung müssen Tutorials gegebenenfalls immer wieder angepasst werden.

Variante	Art	Interaktiv	Vorteil	Nachteil
Text	extern	Nein	schnelle Umsetzung	Nur begrenzter Nutzen möglich
Video	extern	Nein	Erklärung anhand visueller Komponenten	Nicht interaktiv und nur begrenzter Nutzen
Screencast	extern	Ja	Direktes Vorstellen der Nutzung der Anwendung	Ungültig, sobald sich Benutzeroberfläche ändert
Erste Schritte	intern	Ja	Optimale Erklärung durch interaktive Nutzung	Aufwendige Entwicklung und abhängig von Entwicklung der Anwendung
Tutonium	extern	Ja	Interaktive Nutzung ohne Wechsel zwischen Tutorial und Anwendung	Aufwendige Entwicklung mit Limitierungen und ungültig, sobald sich Benutzeroberfläche ändert

Tabelle 3.1: Übersicht der verschiedenen Tutorials.

3.2 Auswahl

Gerade weil QDAcity noch stark weiterentwickelt wird und sich deshalb viele Oberflächen und Funktionen noch verändern werden, stellt sich eine solche Mischform aus integrierten und externen Tutorial als ungeeignet heraus. Wie in der Tabelle 3.1 zu sehen ist, sind die Tutorials bei einer Entwicklung durch das Framework Tutonium zwar nicht in die Entwicklung der Webanwendung integriert, die erstellten Tutorials müssen aber selbst bei kleinen Änderungen angepasst und gepflegt werden. Auch ist das Erstellen der Tutorials keines Wegs trivial und beinhaltet neben technischem Vorwissen auch noch externe Programme.

QDAcity möchte dem Nutzer das Feature anbieten, individuelle Tutorials aus vordefinierten Schritten erstellen zu können. Über die Tutorials soll Wissen vermittelt und Workflows oder Methoden abgebildet werden können. Der Nutzer soll optimal bei der Erstellung unterstützt werden und ohne explizites technisches Wissen das Tutorial System nutzen können. Da der Nutzer somit ein Feature der Anwendung nutzt, ist ein integriertes Tutorial System besser geeignet als ein externes oder eine Mischform. Die Webanwendung selbst kümmert sich um Änderungen innerhalb der Benutzeroberfläche und der Nutzer, der das Tutorial erstellt hat, muss dieses nicht selbst anpassen. Der Nutzer muss für die Erstellung und Verwendung der Tutorials auch die Webanwendung nicht verlassen oder externe Anwendungen nutzen.

4 Anforderungen

4.1 Funktionale Anforderungen

Zunächst werden die Funktionalen Anforderungen (FA) an das Tutorial System definiert.

4.1.1 Vordefinierte Tutorials

FA 1: Das Tutorial System soll Tutorials, die im Vorfeld vom System erstellt wurden und nicht mehr durch den Nutzer erstellt werden müssen, laden und bereitstellen können.

FA 2: Das Tutorial System soll vordefinierte Tutorials für die in Kapitel 2 vorgestellten Bereiche bereitstellen.

4.1.2 Tutorials

FA 3: Das Tutorial System soll vom Nutzer erstellte Tutorials laden und anzeigen können.

FA 4: Der Nutzer soll Tutorials erstellen können.

FA 5: Der Nutzer soll den Namen, die Beschreibung oder die Schritte von selbst erstellten Tutorials ändern können.

FA 6: Der Nutzer soll selbst erstellte Tutorials löschen können.

FA 7: Der Nutzer soll selbst erstellte Tutorials über eine URL mit anderen Nutzern teilen können.

4.1.3 Tutorial Fortschritte

FA 8: Das Tutorial System soll dem Nutzer zu jedem Tutorial seinen Fortschritt anzeigen, falls vorhanden. Dabei soll der Status, die Anzahl der abgeschlossenen und noch zu absolvierenden Schritte angezeigt werden.

FA 9: Das Tutorial System soll erkennen, wenn einzelne Schritte vom Nutzer ausgeführt werden und den Fortschritt des Nutzers aktualisieren, wenn der durchgeführte Schritt dem gerade aktiven Schritt des Tutorials entspricht.

FA 10: Das Tutorial System soll dem Nutzer anzeigen, welcher Schritt des Tutorials durchzuführen ist.

FA 11: Das Tutorial System soll die HTML Elemente optisch hervorheben, die der Nutzer zur Durchführung des gerade aktiven Schrittes benötigt.

FA 12: Das Tutorial System soll dem Nutzer anzeigen, wenn ein Schritt erfolgreich ausgeführt wurde.

FA 13: Das Tutorial System soll dem Nutzer anzeigen, wenn das Tutorial erfolgreich abgeschlossen wurde.

FA 14: Der Nutzer soll Tutorials starten können.

FA 15: Der Nutzer soll aktive Tutorials stoppen können.

FA 16: Der Nutzer soll pausierte Tutorials weiterführen können.

FA 17: Der Nutzer soll seinen Tutorialfortschritt löschen können.

FA 18: Der Nutzer soll seinen Tutorialfortschritt zurücksetzen können.

FA 19: Das Tutorial System soll Fortschritte von Tutorials korrekt zurücksetzen, wenn diese geändert werden.

FA 20: Das Tutorial System soll Fortschritte löschen, wenn das dazugehörige Tutorial gelöscht wird.

4.2 Nicht-Funktionale Anforderungen

Die Nicht-Funktionale Anforderungen (NFA) werden nach dem Produkt Qualitätscharakteristiken der ISO DIN 25010 definiert. Dabei werden, wie in Abbildung 4.1 gezeigt, acht Kriterien mit ihren jeweiligen Unterkategorien unterschieden (ISO/IEC 25010, 2011).

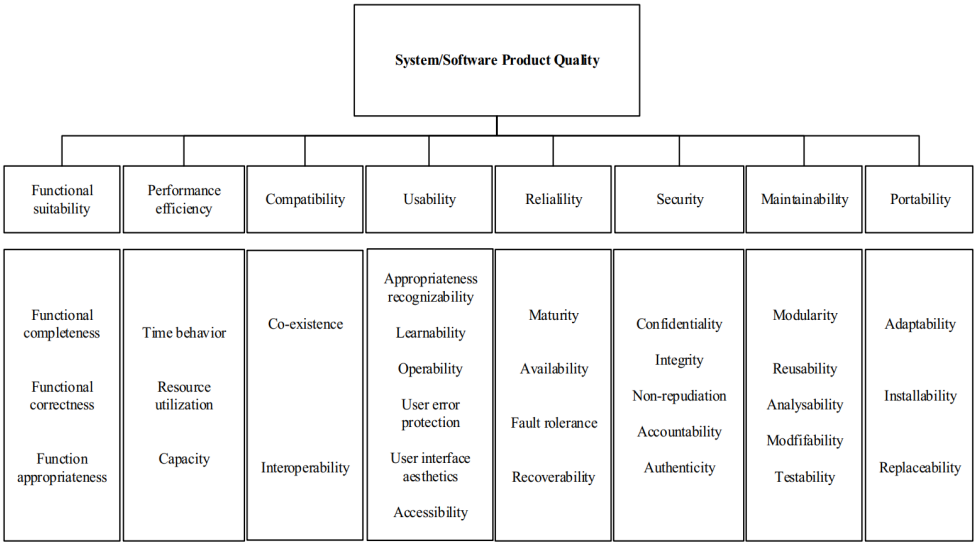

Abbildung 4.1: Software Produkt Qualität Modell der ISO/IEC 25010 (Gong et al., 2016).

4.2.1 Funktionale Eignung

NFA 1: Das Tutorial System soll vollständig funktionstüchtig sein und der Nutzer soll Tutorials nutzen können.

NFA 2: Das Tutorial System soll keine existierende Funktionalität beeinträchtigen.

4.2.2 Effizienz

NFA 3: Das Tutorial System soll den existierenden Cache nutzen, um die bei Anfragen aus der Datenbank geladene Daten zu cachen.

4.2.3 Kompatibilität

NFA 4: Das Tutorial System soll mit den in QDAcity verwendeten Google Diensten (Google App Engine, Google Datastore und Google Cloud Endpoints) kompatibel sein.

NFA 5: Das Tutorial System und die zur Implementierung verwendeten Abhängigkeiten sollen mit den in QDAcity verwendeten Abhängigkeiten kompatibel sein.

4.2.4 Benutzbarkeit

NFA 6: Die Texte des Tutorial Systems sollen entsprechend der Internationalisierungsstrategie von QDAcity übersetzt werden.

NFA 7: Das Farbschema des Tutorial Systems soll mit dem von QDAcity übereinstimmen.

4.2.5 Zuverlässigkeit

NFA 8: Das Tutorial System soll HTTP Statuscodes und in der Benutzeroberfläche Fehlermeldungen zurückgeben.

4.2.6 Sicherheit

NFA 9: API Endpunkte sollen nur Anfragen von angemeldeten Nutzern bearbeiten und im Fehlerfall einen HTTP Statuscode zurückliefern.

NFA 10: Nutzer dürfen nur ihre eigenen Tutorials ändern oder löschen.

NFA 11: Nutzer dürfen nur ihre eigenen oder geteilte Tutorials starten, pausieren, weiterführen und zurücksetzten.

NFA 12: Nutzer dürfen nur auf ihre eigenen Tutorial Fortschritte zugreifen und diese löschen.

4.2.7 Wartbarkeit

NFA 13: Für die hinzugefügte Funktionalität soll eine Codeabdeckung von mehr als 90% der *Lines of Code (LOC)* im Backend durch Unittests und Integrationstests erreicht werden.

NFA 14: Im Frontend sollen alle vordefinierten Tutorials durch E2E Tests getestet werden.

4.2.8 Übertragbarkeit

NFA 15: Das Tutorial System soll das Hinzufügen von neuen Schritten unterstützen.

NFA 16: Das Tutorial System soll die Webbrowser Google Chrome und Mozilla Firefox unterstützen.

5 Architektur

Im folgenden Kapitel wird die Architektur des Tutorial Systems auf Grundlage der Anforderungen aus Kapitel 4 beschrieben.

5.1 Tutorial System

Das Tutorial System soll interaktiv und in die Anwendung integriert sein, sowie dem Nutzer die verschiedenen Funktionen von QDAcity durch Tutorials näher bringen. Da QDAcity durch ein hohes Maß an Komplexität nicht intuitiv verständlich ist, gestaltet sich besonders für neue Nutzer das Erlernen der Funktionalität als schwierig. Im Folgenden werden die Funktionen und die einzelnen Komponenten des Tutorial Systems beschrieben.

5.1.1 Funktionen

Dem Nutzer soll durch das Tutorial System die folgende Funktionalität geboten werden:

- Die Möglichkeit Tutorials zu verwalten
- Die Möglichkeit selbst Tutorials zu erstellen
- Die Möglichkeit Tutorials durchzuführen
- Die Möglichkeit Tutorials zu teilen
- Die Möglichkeit Tutorials von anderen Nutzern durchzuführen

Tutorials verwalten

Der Nutzer soll auf eine Übersicht aller für ihn verfügbaren Tutorials und seiner Fortschritte zu diesen zugreifen können. Die für den Nutzer verfügbaren Tutorials setzen sich dabei aus seinen selbst erstellten Tutorials (User Tutorials), den vom Tutorial System vordefinierten Tutorials (Basis Tutorials) und den durch Einladungslinks gestarteten Tutorials von anderen Nutzern zusammen. Die Basis

Tutorials werden von QDAcity zur Verfügung gestellt und sollen den Nutzern verschiedene Funktionen in QDAcity nahebringen.

Die Abbildung 5.1 zeigt, dass der Nutzer neben dem Zugriff auf die verschiedenen Tutorials auch unterschiedliche Aktionen ausführen kann. So sollen sowohl einzelne Tutorials gestartet als auch gestoppt werden können. Der Fortschritt des Nutzers bei einzelnen Tutorials soll zurückgesetzt oder ganz gelöscht werden können. Je nach Berechtigung soll der Nutzer Tutorials auch ändern oder löschen können, wobei dann berücksichtigt werden muss, dass andere Nutzer die Tutorials auch gestartet haben könnten und die Fortschritte dieser Nutzer dann zurückgesetzt oder gelöscht werden müssten. Dies ist nötig, da sich QDAcity stetig verändert und weiterentwickelt wird und was dazu führt, dass einzelne Schritte der Tutorials womöglich nicht mehr in der Anwendung verfügbar sind oder sich Abläufe innerhalb der Anwendung verändert haben und ganz neue Schritte hinzugekommen sind. Dadurch könnten die Tutorials nicht mehr durchführbar sein und nur durch Anpassungen wieder nutzbar werden.

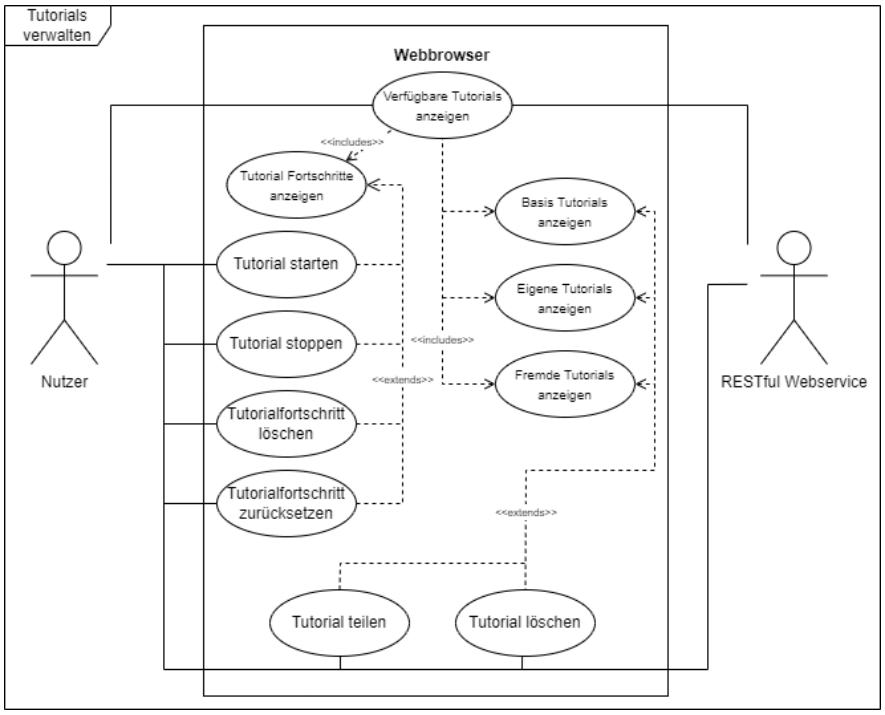

Abbildung 5.1: Use-Case in dem die Tutorials verwaltet werden.

Tutorial erstellen

Wie in Abbildung 5.2 gezeigt, kann der Nutzer Tutorials erstellen, wobei dieser dem Tutorial einen Namen, eine Beschreibung und Schritte hinzufügen muss. Die einzelnen Schritten können dabei optional auch mit einem Bearbeitungsvermerk

versehen werden. Da bei der Entwicklung von QDAcity Internationalisierung (i18n) eine wichtige Rolle spielt, soll das Tutorial System Mehrsprachigkeit unterstützen. So können die Nutzer Beschreibungen des Tutorials und die Bearbeitungsvermerke jeweils in den unterstützten Sprachen hinzufügen.

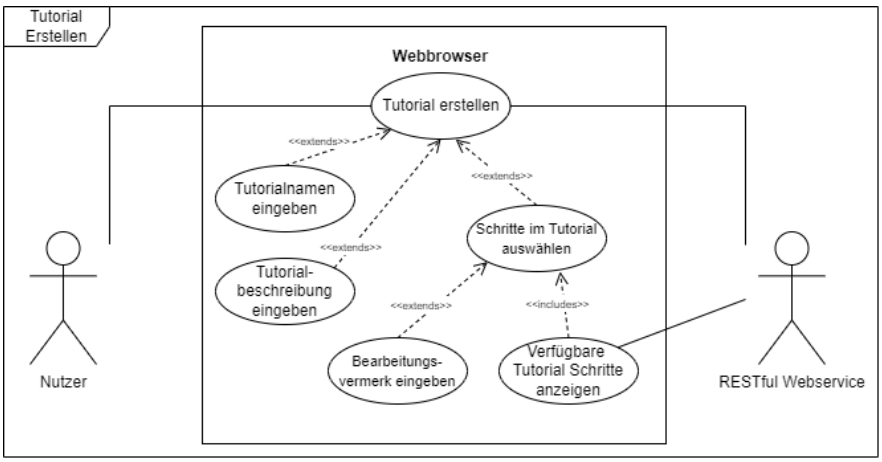

Abbildung 5.2: Use-Case in dem ein Tutorial erstellt wird.

Des Weiteren wird dem Nutzer bei der Erstellung des Tutorials viel Flexibilität geboten, da nicht vorhergesehen werden kann, wie genau der Prozess aussieht, der durch das Tutorial beschrieben wird. So können Nutzer einzelne Schritte nicht nur einmal sondern beliebig oft in Tutorials verwenden, wobei auch keine Vorbedingungen für einzelne Schritte definiert wurde. Dies wird beim Zugriff auf Projekte gut ersichtlich, da zwar zuerst ein Projekt erstellt werden muss, um darauf zugreifen zu können, jedoch können Nutzer auch zu bereits existierenden Projekten hinzugefügt werden, wodurch nicht erst ein Projekt erstellt werden muss. Der Nutzer trägt die Verantwortung, ob das von ihm erstellte Tutorial inhaltlich korrekt ist.

Tutorials durchführen

Sobald ein Tutorial aktiv ist, können Nutzer dessen einzelnen Schritte durchlaufen, wie in Abbildung 5.3 aufgezeigt. Dabei soll der Fortschritt des Tutorials nur auf den nächsten Schritt aktualisiert werden, wenn der Nutzer die für den Schritt benötigte Handlung ausgeführt hat. Zudem soll auf der Benutzeroberfläche spezifisch für den Schritt hervorgehoben werden, welche Elemente für diesen Schritt wichtig sind oder auf welche Elemente der Nutzer beispielsweise klicken muss. Für jeden Schritt soll dessen Beschreibung in der Sprache des Nutzers und die beim Erstellen definierten Bearbeitungsvermerke angezeigt werden. Der Nutzer kann die Durchführung des Tutorials jederzeit pausieren und wieder starten. Nach erfolgreichem Durchlaufen des Tutorials wird dem Nutzer dies in der Übersicht der Tutorials durch den Status des Tutorials angezeigt.

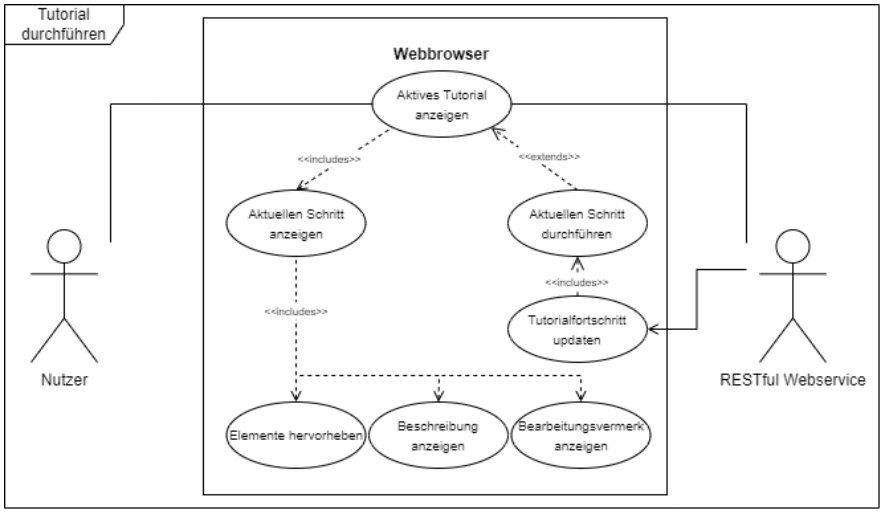

Abbildung 5.3: Use-Case in dem ein Tutorial durchgeführt wird.

Tutorials teilen

Nutzer können Tutorials, die von ihnen selbst erstellt wurden, mit anderen Nutzern über einen Link teilen. Der Ersteller des Tutorials hat Zugriff auf den Einladungslink und kann diesen Link jederzeit invalidieren, falls der Zugriff auf das Tutorial gestoppt werden soll. Andere Nutzer können über den Einladungslink das Tutorial starten und dieses durchlaufen. Nachdem ein Tutorial über einen solchen Link gestartet wurde, wird es auch in der Übersicht aller verfügbaren Tutorials des Nutzers aufgelistet.

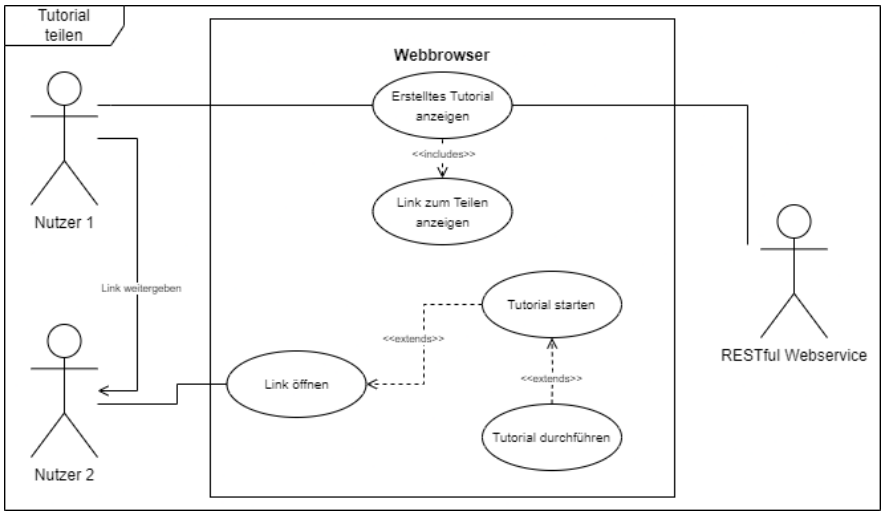

Abbildung 5.4: Use-Case in dem ein Tutorial mit einem anderen Nutzer geteilt wird.

5.1.2 Backend

Da das Tutorial System im Backend aus verschiedenen Komponenten besteht, soll in diesem Kapitel eine abstrakte Übersicht über jene gegeben werden. Die genaue Implementierung der Komponenten und des darunterliegenden Datenmodells wird in Kapitel 6 beschrieben.

Tutorial System

Im Backend ist das Tutorial System für das Bereitstellen, Verwalten und Speichern der benötigten Daten über CRUD Operationen zuständig, um so die in 5.1.1 definierten Funktionen bieten zu können. Dafür wurden für die Daten einzelne Komponenten gebildet, die im Folgenden beschrieben werden.

Das Tutorial System trennt zunächst die Tutorials und die Fortschritte zu den einzelnen Tutorials in separate Komponenten, *Tutorial* und *TutorialState*. *Tutorial* beinhaltet dabei Informationen über sich selbst und eine Liste an Schritten, die selbst als Komponente *TutorialStep* modelliert werden. Die *TutorialState* Komponente beinhaltet Daten über den Nutzer, das Tutorial, den aktuell auszuführenden Schritt und den Status (aktiv/pausiert/beendet) des Tutorials. Die *TutorialStep* Komponenten veranschaulichen dabei Handlungen (*Step*), die vom Nutzer in QDAcity durchgeführt werden sollen, und beinhalten ihre Position in der Liste der durchzuführenden Schritte. Zudem kann jedes *Tutorial* über einen *TutorialLink* geteilt werden.

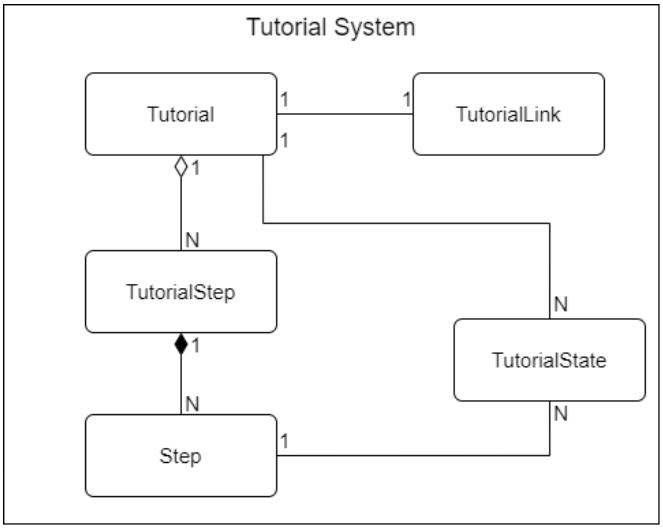

Abbildung 5.5: ER Diagramm der einzelnen Komponenten des Tutorial Systems.

Eine weitete Unterscheidung erfolgt dahingehend, dass die Tutorials einerseits vom Nutzer erstellt werden können und andererseits bereits vordefinierte Basis

Tutorials von System bereitgestellt werden. Die vordefinierten Basis Tutorials sollen dabei die in QDAcity existierende Funktionalität allgemein abdecken und sind für alle Nutzer verfügbar. Die Unterschiede zwischen den beiden Arten werden in Tabelle 5.1 präsentiert.

	Zugriff	**Ersteller**	**Veränderbar**	**Teilbar**
Basis Tutorials	Alle Nutzer	QDAcity	Nein	Nein
Nutzer Tutorials	Ersteller	Nutzer	Ja	Ja

Tabelle 5.1: Unterschiede von vordefinierten Basis Tutorials und den vom Nutzer definierten Tutorials innerhalb des Tutorial Systems.

REST Endpunkte

Damit die Frontend Komponente des Tutorials Systems mit der Backend Komponente kommunizieren kann, werden von diesem REST Endpunkte bereitgestellt, an die das Frontend HTTP Anfragen senden kann. Dies unterscheidet sich von den restlichen Endpunkten in QDAcity, da viele dieser Endpunkte eher als Remote Procedure Call (RPC)[1] fungieren. Bei Representational State Transfer (REST)[2] wird der Fokus darauf gelegt, dass es keine spezifischen Methoden oder Aktionen gibt sondern nur Ressourcen, auf welche die HTTP Methoden *GET*, *POST*, *PUT* und *DELETE* ausgeführt werden können. Zudem gibt es keine Session oder andere Zustände, die serverseitig gespeichert werden.

Ressourcen	**Genutzte Methoden**
Tutorial	GET, POST, PUT, DELETE
TutorialState	GET, POST, PUT, DELETE
Step	GET
TutorialLink	GET, POST, DELETE

Tabelle 5.2: REST Ressourcen des Tutorial Systems.

Caching

QDAcity verwendet im Backend bereits den von GAE bereitgestellten Memcache[3], in dem Daten als Key Value Paare gespeichert werden können. Die im Cache gespeicherten Daten müssen nicht erst aus der Datenbank geladen werden und somit sind Zugriffe auf den Memcache deutlich effizienter. Alle in 5.1.2 definierten Ressourcen des Tutorial Systems werden nach dem ersten Zugriff gecached,

[1] https://www.datacenter-insider.de/was-ist-ein-remote-procedure-call--rpc-a-712873/
[2] https://www.opc-router.de/was-ist-rest/
[3] https://cloud.google.com/appengine/docs/legacy/standard/java/memcache

bei Veränderungen der Memcacheeintrag aktualisiert und bei Löschung wird der Memcacheeintrag invalidiert.

Dies ist besonders wirksam bei den Ressourcen *Tutorial* und *Step*, da bei diesen nicht jeder Nutzer auf andere Einträge zugreift, sondern eine Vielzahl der Nutzer den gleichen Eintrag laden möchten. So gibt es bei den Tutorials die Basis Tutorials, welche nur geladen werden und sich nicht ändern können. Das Gleiche gilt für die *Steps*, da diese im System fest definiert sind und kein REST Endpoint existiert, der das Erstellen, Ändern oder Löschen von Steps Nutzern ermöglicht. Somit kann hier nach dem ersten Laden der Einträge bei jeder weiteren Anfrage Zeit und Ressourcen gespart werden.

Systemübersicht

Abschließend wird eine Übersicht des Systems im Backend gegeben (siehe Abbildung 5.6), die aus den REST Endpoints, der Datenbankschnittstelle, dem Memcache und der Tutorial System Komponente, welche sich um die CRUD Operationen kümmert, bestehen.

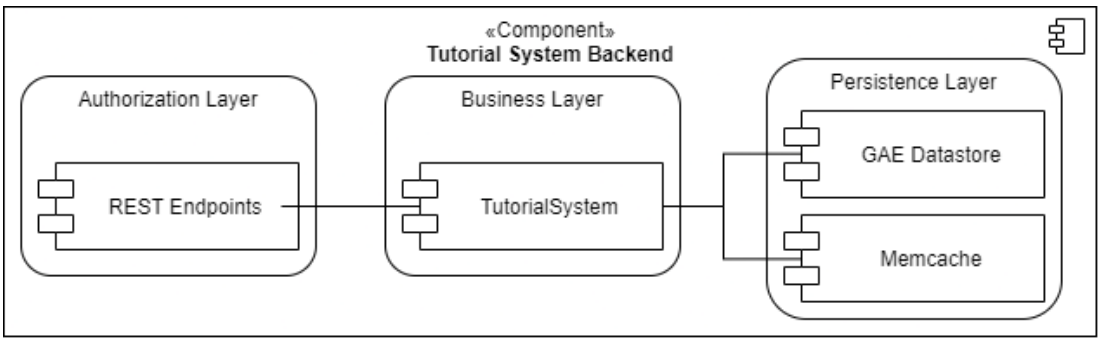

Abbildung 5.6: Übersicht des Tutorial Systems im Backend.

Dabei können drei Schichten unterschieden werden:

- Authorization Layer
- Business Layer
- Persistence Layer

Der Authorization Layer und die sich darin befindenden REST Endpoints haben die Aufgabe sicherzustellen, dass nur autorisierte, eingeloggte Nutzer an die Endpunkte Requests schicken.

Der Business Layer beinhaltet jegliche implementierte Logik zu dem Tutorial System und ist für Durchführung der CRUD Operationen zuständig.

Das Persistence Layer übernimmt die Speicherung der Daten und bietet Schnittstellen für den Zugriff auf diese an. In QDAcity existieren, wie bereits in 2.2.2

und 5.1.2 beschrieben, mit dem GAE Datastore und dem Memcache zwei Komponenten im Persistence Layer.

5.1.3 Frontend

Das Tutorial System ist im Backend hauptsächlich für CRUD Operationen zuständig, im Frontend muss das Tutorial System vor allem erkennen, wann der Nutzer bestimmte Handlungen durchführt, um so zu registrieren, ob der *TutorialState* auf den nächsten *Step* aktualisiert werden muss. Abbildung 5.7 zeigt die Funktionsweise des Tutotrial Systems in Frontend und wie dort zentral der Zustand der für den Nutzer verfügbaren *Tutorials*, *Steps*, *TutorialStates* und *TutorialLinks* verwaltet wird und das System diesen Zustand den UI Komponenten bereitstellt, die Informationen über das Tutorial anzeigt oder ausgewählte Elemente für Schritte hervorhebt. Das Frontend kümmert sich auch um Internationalisierung, da die Tutorials einige Texte beinhalten, die je nach Nutzer in die passenden Sprache übersetzt werden müssen und erst dann angezeigt werden können.

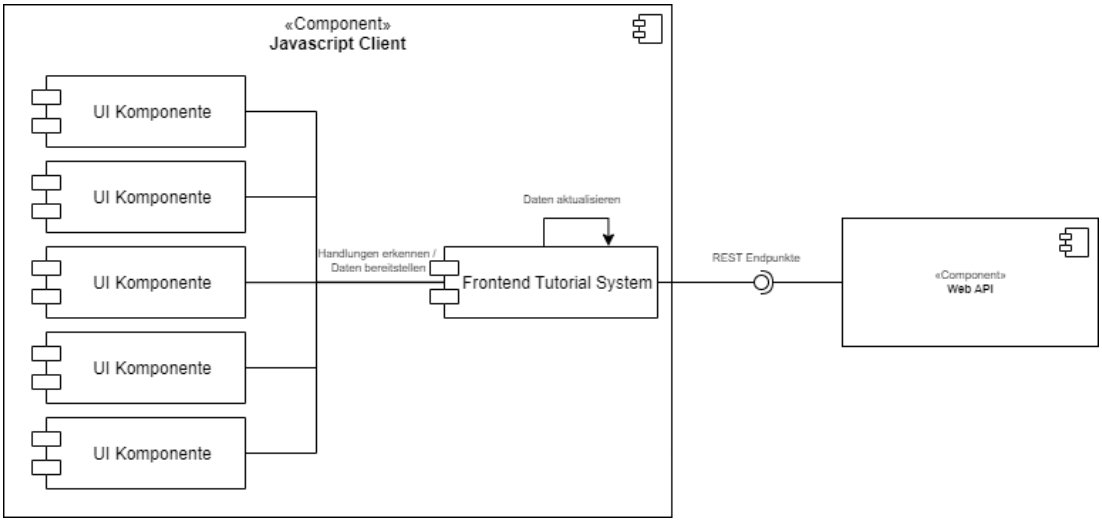

Abbildung 5.7: Übersicht des Tutorial Systems im Frontend.

Um diese Aufgaben im Frontend realisieren zu können, werden Software Design Pattern vorgestellt, die zentrale Eigenschaften anbieten, die bei der Implementierung benötigt werden. Ob ein Einsatz dieser Pattern in React möglich und sinnvoll ist, wird in Kapitel 6 beschrieben.

Singleton

In objektorientierten Sprachen wird das Singleton Pattern verwendet, um sicherzustellen, dass nur eine Instanz einer Klasse verfügbar ist (Musch, 2021b). Diese

Eigenschaft wird im Tutorial System zur Verwaltung der Daten benötigt, da diese Verwaltung gekapselt an nur einer einzigen Stelle stattfinden soll. Dies hat den Vorteil, dass sich die einzelnen UI Komponenten nicht darum kümmern müssen und auch keine weiteren Maßnahmen nötig sind, um die Konsistenz der Daten in den UI Komponenten zu gewährleisten. Zudem gestaltet sich dieser Ansatz als sehr effizient und ressourcenschonend, da sich die Anzahl der HTTP Requests zum Backend auf ein Minimum reduziert.

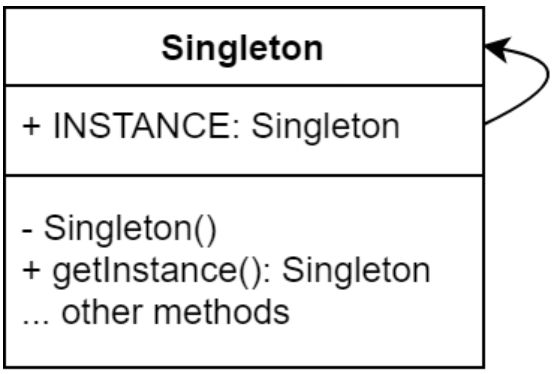

Abbildung 5.8: UML-Diagramm des Singleton Patterns.

Observer

Zentrale Aufgabe des Tutorial Systems im Frontend ist es, zu erkennen, ob bestimmte Handlungen vom Nutzer durchgeführt werden. Zunächst könnte das Tutorial System stetig die Handlungen des Nutzers überwachen und prüfen, ob der aktuelle Schritt des aktiven Tutorials erfüllt wurde. Dieser Ansatz ist aber sehr aufwendig und ineffizient, weshalb Eigenschaften des Observer Patterns genutzt werden könnten.

Das Observer Pattern setzt das Prinzip von *Publish /Subscribe* um, bei dem ein *Subscriber* nicht ständig selbst nach neuen Ereignissen schaut, sondern sich vom *Pubisher* benachrichtigen lässt (Musch, 2021a). Dieses Prinzip würde den Aufwand im Tutorial System enorm verringern, wenn das Tutorial System als *Subscriber* nur dann von den UI Komponenten (*Publisher*) benachrichtigt wird, wenn bestimmte Handlungen ausgeführt wurden.

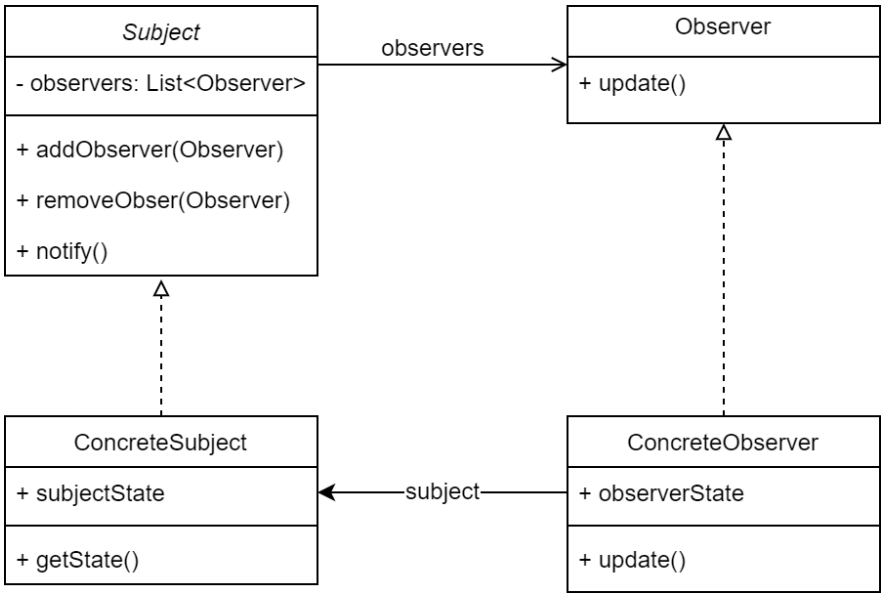

Abbildung 5.9: UML-Diagramm des Observer Patterns.

6 Implementierung

In diesem Kapitel wird die Implementierung des Tutorial Systems auf Grundlage der Architektur, die in Kapitel 5 vorgestellt wurde, beschrieben. Das Ziel der Implementierung ist die Erfüllung der Anforderungen aus Kapitel 4. Die Technologien, mit denen die einzelnen Komponenten umgesetzt werden, stimmen mit den Technologien überein, die von QDAcity zur Implementierung genutzt werden. So handelt es sich bei QDAcity um eine Google App Engine Anwendung, die mit Java 8 umgesetzt wurde. Im Frontend wurde JavaScript (JS) und die Bibliothek React eingesetzt.

6.1 Backend

Die Beschreibung der serverseitigen Implementierung des Tutorial Systems setzt sich aus der Definition eines passenden Datenmodells für das Tutorial System, der Implementierung der Businesslogik, den REST Endpunkten und der Initialisierung der vom System vordefinierten Basis Tutorials und Schritten zusammen.

6.1.1 Datenmodell

Zunächst wird für die Daten, die vom Tutorial System benötigt und verwendet werden, ein Datenmodell erstellt. Dieses basiert auf den einzelnen Komponenten aus Kapitel 5.1.2, wie in Abbildung 5.5 dargestellt. Die im Datenmodell verwendeten Entitäten müssen dabei so konzipiert sein, dass diese von JDO zur Speicherung verwendet werden können.

Dies erfolgt zum einen durch von JDO bereitgestellte Annotationen und zum anderen muss jede Entität das Interface Serializable implementieren. Durch die Klassen-Annotation `@PersistenceCapable` kann eine Java Klasse als JDO konforme Entität gekennzeichnet werden. Zudem kann jede Variable einer Entität, die als Feld in der Datenbank gespeichert werden soll, mit `@Persistent` versehen werden. Durch `@PrimaryKey` wir der Primärschlüssel definiert, über welchen der eindeutige Zugriff erfolgt.

Die Entitäten wurden wie in Abbildung 6.1 definiert, die einen Überblick über diese, ihre Attribute und ihre Beziehungen zueinander gibt. Das Datenmodell basiert dabei auf der Abbildung 5.5 und die einzelnen Klassen werden in den nachfolgenden Kapiteln genauer erläutert.

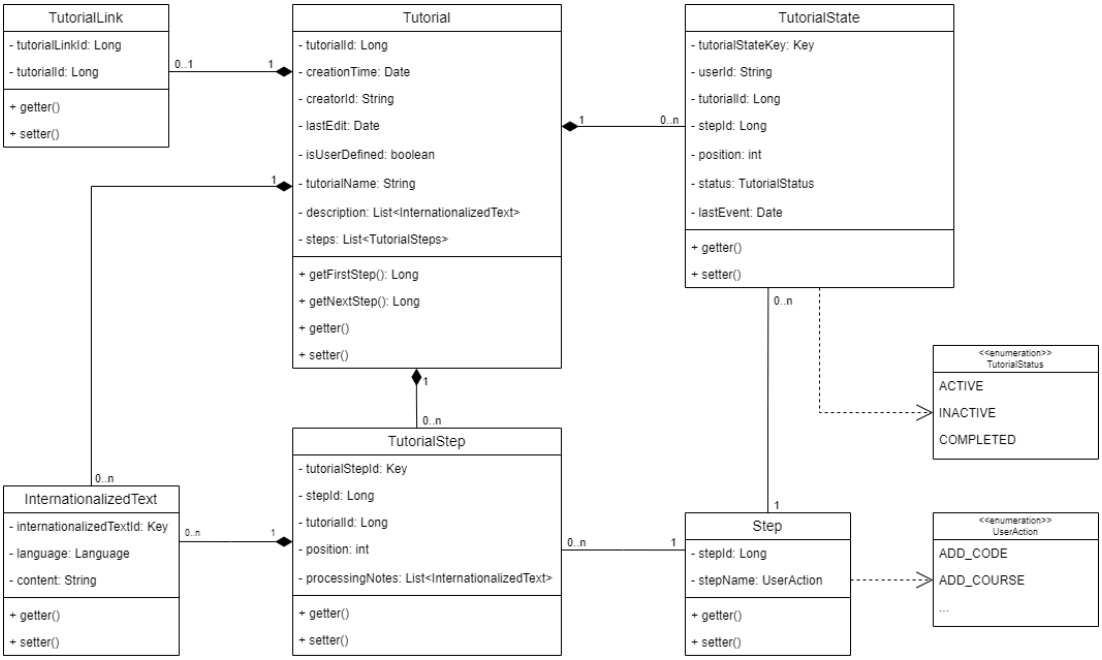

Abbildung 6.1: Datenmodell des Tutorial Systems.

Tutorial

Die `Tutorial` Entität stellt den zentralen Bestandteil des Tutorial Systems dar und modelliert dabei nicht die Daten, die beim Starten oder Durchführen eines Tutorials erstellt oder verwendet werden, sondern dient als Vorlage für die Fortschritte der Nutzer. Das `Tutorials` beinhaltet, wie in Tabelle 6.2 zu sehen ist, vor allem die durchzuführenden Schritte, aber auch seinen Namen, eine Beschreibung, den Zeitpunkt der Erstellung und der letzten Änderungen und die ID des Erstellers.

Neben den Attributen besitzt das `Tutorial` auch die Methoden `getFirstStep()` und `getNextStep()`, die bei der Aktualisierung der `TutorialStates` verwendet werden (siehe Tabelle 6.1).

Methode	Rückgabewert	Typ	Ausnahmen
getFirstStep	Der erste Schritt des Tutorials	Long	`IllegalStateException` für ein Tutorial ohne Schritte
getNextStep	Der nächste Schritt anhand der ID und Position des aktuellen Schrittes	Long	`IllegalArgumentException` bei ungültigen Parametern und `TutorialStateAlreadyAtLastStepException` falls der letzte Schritt erreicht wurde

Tabelle 6.1: Methoden der Tutorial Entität.

Tutorial		
Attribute	Typ	Bedeutung
tutorialId	Long	ID als Primärschlüssel des Tutorials, über den dieses eindeutig identifizierbar ist
creationTime	Date	Zeitpunkt der Erstellung
creatorId	String	ID des Erstellers
lastEdit	Date	Zeitpunkt der letzten Anpassung
isUserDefined	boolean	Legt fest, ob das Tutorial von einem Nutzer erstellt wurde oder es ein Basis Tutorial ist
tutorialName	String	Name des Tutorials
description	List<InternationalizedText>	Beschreibung des Tutorials in verschiedenen Sprachen
steps	List<TutorialStep>	Liste der Schritte

Tabelle 6.2: Attribute der Tutorial Entität mit Typ und Bedeutung.

TutorialState

Die `TutorialState` Entität speichert den Fortschritt zu den `Tutorials`, weshalb nur ein `TutorialState` pro `Tutorial` für jeden Nutzer erlaubt ist. Gespeichert wird dabei welcher Nutzer welches Tutorial durchführt, bei welchem Schritt dieser gerade ist und welchen Zustand der `TutorialState` hat (siehe Tabelle 6.4).

Für die Darstellung des Zustands wurde das Enum `TutorialStatus`, wie in Tabelle 6.3 beschrieben, verwendet.

Der `TutorialStatus` ist von großer Bedeutung, da sich die Benutzeroberfläche für Nutzer nur mit aktivem Tutorial verändert und Nutzer maximal ein Tutorial aktiv verwenden dürfen.

TutorialStatus	
Wert	Bedeutung
ACTIVE	Das Tutorial wird gerade aktiv durchgeführt
INACTIVE	Die Durchführung des Tutorials wurde pausiert
COMPLETED	Der Nutzer hat das Tutorial vollständig durchgeführt

Tabelle 6.3: Attribute der Tutorial Entität mit Typ und Bedeutung.

TutorialState		
Attribute	Typ	Bedeutung
tutorialStateKey	Key	ID und Primärschlüssel des TutorialStates, über den dieses eindeutig identifizierbar ist
userId	String	ID des Nutzers, zu dem der Fortschritt des Tutorials gehört
tutorialId	Long	ID des Tutorials
stepId	Long	ID des Schrittes, der aktuell durchzuführen ist
position	int	Position des Schrittes in der Liste der Schritte des Tutorials
status	TutorialStatus	Status des Tutorials
lastEvent	Date	Zeitpunkt der letzten Aktualisierung

Tabelle 6.4: Attribute der TutorialState Entität mit Typ und Bedeutung.

Step

Alle Aktionen die als Schritte in Tutorials durchführbar sind, werden als `Step` Entität dargestellt, um diese in der Datenbank speichern zu können.

Die Aktionen selbst werden als `UserAction` in Form eines Enums dargestellt und beinhalten 51 Werte.

Step		
Attribute	Typ	Bedeutung
stepId	Long	ID und Primärschlüssel des Steps, über den dieses eindeutig identifizierbar ist
name	UserAction	Name der Aktion, die vom Nutzer durchgeführt werden soll

Tabelle 6.5: Attribute der Step Entität mit Typ und Bedeutung.

TutorialStep

Die `TutorialStep` wird benötigt, da im `Tutorial` nicht nur eine Liste an `Steps` gespeichert wird, sondern neben der Position des Schrittes in der Liste auch vom Ersteller definierte Bearbeitungsanmerkungen in verschiedenen Sprachen gespeichert werden sollen.

TutorialStep		
Attribute	Typ	Bedeutung
tutorialStepId	Key	ID und Primärschlüssel des TutorialSteps, über den dieses eindeutig identifizierbar ist
stepId	Long	ID des Schrittes, der die durchzuführende Aktion definiert
tutorialId	Long	ID des Tutorials
position	int	Position des Schrittes, wobei diese nur einmal in der Liste im Tutorial vorkommen darf
processingNotes	List<InternationalizedText>	Bearbeitungsvermerk des Erstellers, der in verschiedenen Sprachen gespeichert werden kann

Tabelle 6.6: Attribute der TutorialStep Entität mit Typ und Bedeutung.

Für die Speicherung der `processingNotes` wird wie für die `description` im `Tutorial` die `InternationalizedText` Entität verwendet. Diese ist nötig, um die Internationalisierung von Inhalten zu unterstützen, die erst vom Nutzer erstellt werden.

InternationalizedText		
Attribute	Typ	Bedeutung
internationalizedTextId	Key	ID und Primärschlüssel, über den dieser eindeutig identifizierbar ist
language	Language	Sprache des Textes
content	String	Der zu speichernde Text

Tabelle 6.7: Attribute der InternationalizedText Entität mit Typ und Bedeutung.

TutorialLink

Um `Tutorials` teilen zu können, wurde die `TutorialLink` Komponente definiert. Über die ID des `TutorialLink` kann das zugehörige Tutorial gestartet werden, ohne dass der Nutzer andere Nutzer direkt einladen muss oder aber Tutorials direkt für alle Nutzer offen sind. Dabei kann durch das Löschen der `TutorialLinks` der Zugang zu den Tutorials auch wieder entzogen werden.

TutorialLink		
Attribute	Typ	Bedeutung
tutorialLinkId	Long	ID und Primärschlüssel, über den dieser eindeutig identifizierbar ist
tutorialId	Long	ID des zugehörigen Tutorials

Tabelle 6.8: Attribute der TutorialLink Entität mit Typ und Bedeutung.

6.1.2 REST Endpunkte

Neben dem Datenmodell wurden eine Reihe von REST Endpunkten mit Hilfe von Google Cloud Endpoints für die verschiedenen Ressourcen im Tutorial System implementiert. Alle Endpunkte wurden in der Klasse `TutorialEndpoint` gekapselt und haben die folgenden Aufgaben:

- REST Endpunkt für das Frontend definieren und bereitstellen
- Autorisierung des Nutzers
- Aufruf der benötigten Businesslogik

Codebeispiel 1 zeigt auf, dass durch die `@ApiMethod` Annotation eine Methode als Endpunkt gekennzeichnet werden kann. Die Endpunkte bestehen dabei nur aus zwei Aufrufen.

Zunächst wird versucht, über das von Google Cloud Endpoints genutzte Userobjekt, den in QDAcity verwalteten User zu erhalten. Sollte dies nicht erfolgreich sein, da es sich nicht um einen angemeldeten Nutzer handelt, wird eine Exception geworfen. Daraufhin wird eine `Controller` Klasse aufgerufen, die die Businesslogik implementiert hat.

```
1  @ApiMethod(name = "tutorial.getUserTutorials",
2          httpMethod = "GET",
3          path = "tutorials/userTutorials")
4  public List<Tutorial> getUserTutorials(User user)
5    throws UnauthorizedException {
6     com.qdacity.user.User qdacityUser = userEndpoint
7                                         .getCurrentUser(user);
8     return tutorialController.getTutorialsOfUser(qdacityUser.getId());
9  }
```

Codebeispiel 1: REST API Endpunkt für das Laden aller Tutorials, die ein Nutzer erstellt hat.

Es werden insgesamt 23 REST Endpunkte für die vier Ressourcen *Tutorial*, *TutorialState*, *Step* und *TutorialLink* angeboten. Dabei wird je nach CRUD Operation eine der HTTP Methoden `GET/POST/PUT/DELETE` verwendet, um die in Kapitel 5.1.1 beschriebenen Funktionen umzusetzen. In Abbildung 6.2 werden die Endpunkte dargestellt, mit denen Tutorials erstellt, verändert und gelöscht werden können. Mit den Endpunkten können Tutorials auch einzeln über ihre ID oder mehrere in Form einer der drei für den Nutzer verfügbaren Arten geladen werden.

- /basicTutorials: alle vom System vordefinierten Tutorials

- /userTutorials: alle vom Nutzer erstellten Tutorials
- /importedTutorials: alle von anderen Nutzern erstellte Tutorials, die durch einen Link gestartet wurden

Abbildung 6.2: REST Endpunkte für die Tutorial Ressource.

Bei der Löschung von Tutorials muss beachtet werden, dass alle zu diesem gehörenden *TutorialStates* ebenfalls gelöscht werden.

Besonders viele Endpunkte wurden für die *TutorialState* Ressource implementiert, da diese komplex ist und viel Funktionalität beinhaltet. *TutorialStates* können über die ID des *Tutorials* oder der ID eines *TutorialLinks* erstellt werden, je nachdem ob es sich um ein Basis Tutorial, das Tutorial des Nutzers selbst oder ein Tutorial eines anderen Nutzers handelt. Auch kann der *TutorialStatus*, wie in 6.3 beschrieben, aktualisiert werden.

Ein wichtiger Bestandteil der *TutorialState* Ressource ist der Schritt, den der Nutzer aktuell durchführen soll. Dieser Schritt kann über Endpunkte entweder bei erfolgreicher Durchführung des aktuellen Schrittes auf den nächsten Schritt aktualisiert oder auf den ersten Schritt des Tutorials zurückgesetzt werden, um so das Tutorial von Anfang an durchzuführen.

TutorialStates können nach verschiedenen Kriterien, wie der eigenen ID, der ID des zugehörigen Tutorials, des aktuellen Nutzers oder das gerade aktive *TutorialState* geladen werden.

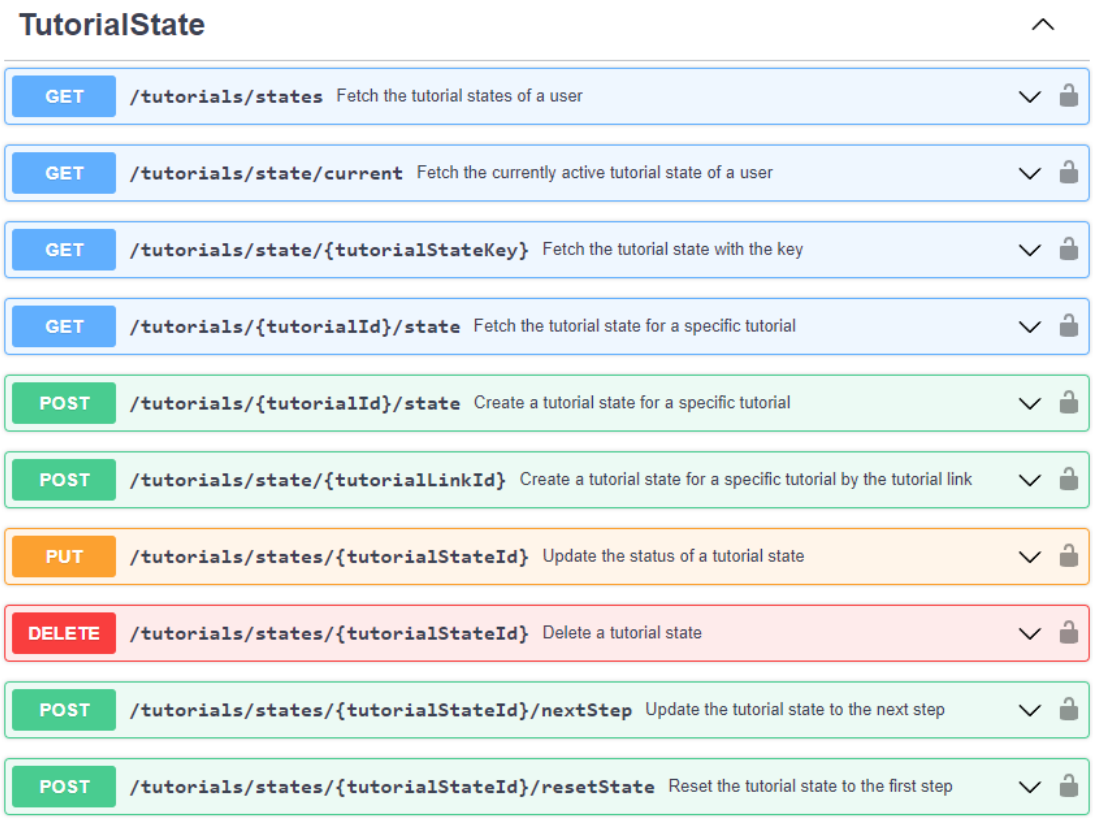

Abbildung 6.3: REST Endpunkte für die TutorialState Ressource.

Anders als die restlichen Ressourcen können *Steps* nur geladen und nicht erzeugt, verändert oder gelöscht werden. Dabei können einzelne oder alle *Steps* geladen werden. Die Anzahl der Steps kann nur durch das Tutorial System geändert werden.

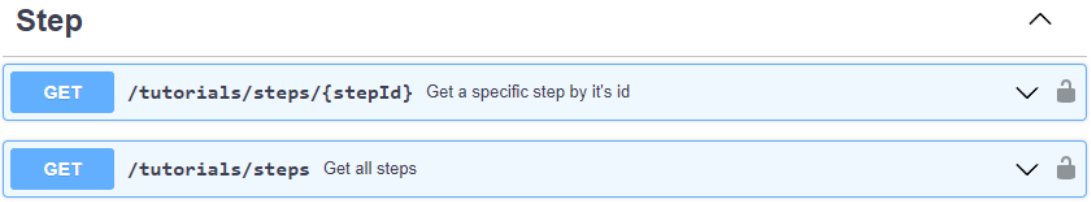

Abbildung 6.4: REST Endpunkte für die Step Ressource.

TutorialLinks werden zum Teilen von Tutorials erstellt und genutzt. Nur die Ersteller eines Tutorials können einen *TutorialLink* für dieses erstellen und durch das Löschen von diesem anderen Nutzern den Zugang zum Tutorial entziehen.

Abbildung 6.5: REST Endpunkte für die TutorialLink Ressource.

6.1.3 Businesslogik

Die Businesslogik des Tutorial Systems wird in `Controller` Klassen implementiert und für die verschiedenen Ressourcen in jeweils einem `Controller` gekapselt (siehe Abbildung 6.6). Dies unterscheidet sich von der generellen Implementierung im Backend von QDAcity, da dort keine Aufteilung der Implementierung existiert. Die Methode, die den Endpunkt darstellt, beinhaltet meist bereits den ganzen Code für Autorisierung, Datenbankzugriffe und Error-Handling. Somit haben die Methoden mehr als nur eine Aufgabe und der Code ist auch kaum wiederverwendbar.

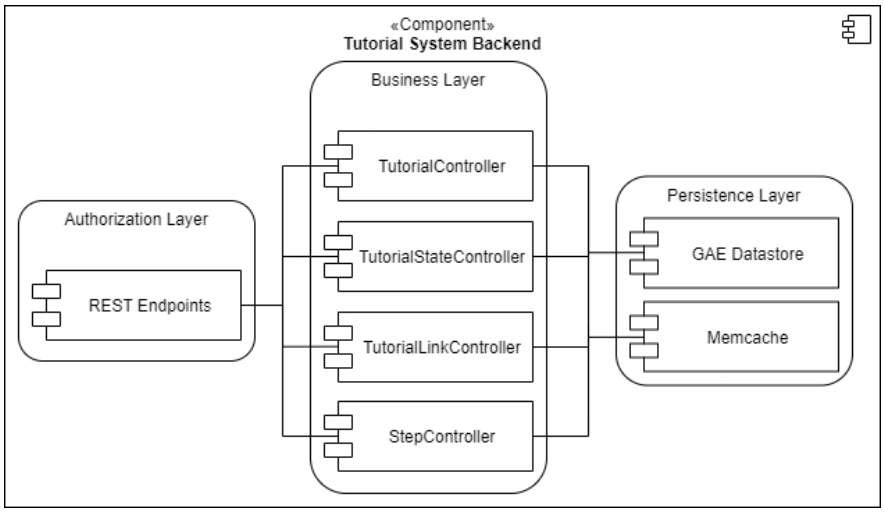

Abbildung 6.6: Überblick über das Tutorial System Backend und die Businesslogik.

Da im Tutorial System Code wiederverwendet werden soll, werden Aufgaben in einzelne Methoden gekapselt und dabei besonderen Wert darauf gelegt, dass diese nur eine Aufgabe haben und so schlank wie möglich bleiben.

Die Methoden können dabei allgemein in drei Kategorien differenziert werden:

- Öffentliche Methoden, die die Funktionen der Businesslogik darstellen
- Aufrufe der Datenbankschnittstellen
- Assertions und Error-Handling

Öffentliche Methoden

Die öffentlichen Methoden der `Controller` sind die Schnittstellen, die von den Endpunkten verwendet werden und implementieren nicht direkt die CRUD Operationen, sondern rufen diese auf und geben das Ergebnis zurück. Zudem führen diese Methoden Checks aus und werfen gegebenenfalls eine Exception.

```
1  public TutorialState createTutorialState(User user, Long tutorialId) {
2          Tutorial tutorial = controller.loadTutorialById(tutorialId);
3      Authorization.checkAuthorization(user, tutorial);
4          throwIfTutorialStateAlreadyExists(user, tutorialId);
5          throwIfUserHasAnActiveTutorialState(user);
6          return insertTutorialState(user, tutorial);
7  }
```

Codebeispiel 2: Öffentliche Methode zum Erstellen eines *TutorialState*.

Die im Codebeispiel 2 gezeigte Methode ruft hierbei die Funktion `insertTutorial()` auf, die direkt die Datenbankschnittstelle aufruft und einen neuen *TutorialState* in die Datenbank einfügt.

Datenbankschnittstellen

Das Laden, Speichern und Löschen von Daten in der Datenbank wurde in eigene Methoden gekapselt, um diese leichter wiederverwendbar zu machen. JDO stellt dabei den `PersistenceManager` und `Queries` bereit und je nach Aufruf, muss eine passende Query definiert werden, die dann vom `PersistenceManager` ausgeführt wird.

Dadurch, dass die öffentlichen Methoden für das Durchführen von Checks zuständig sind und die Implementierung der CRUD Operationen nur aufgerufen wird, wenn keine Probleme aufgetreten sind, kann in diesen vollständig auf Checks verzichtet und redundanter Code vermieden werden.

```
1  private List<TutorialState> loadAllTutorialStatesOfUser(User user) {
2   PersistenceManager persistenceManager = getPersistenceManager();
3   Query query = persistenceManager
4    .newQuery(TutorialState.class, "userId == :userIdParameter");
5   try {
6    return (List<TutorialState>) query.execute(user.getId());
7   } finally {
8    persistenceManager.close();
9   }
10 }
```

Codebeispiel 3: Aufruf der Datenbankschnittstelle, um alle *TutorialStates* eines Nutzers zu laden.

Error-Handling

Die API gibt in Form von HTTP Codes an den Nutzer zurück, ob ein Fehler aufgetreten ist oder nicht. Dabei kann durch den Code direkt erkannt werden, welches Szenario eingetreten ist. In den Controllern werden die Methoden, die als Checks dienen und eine Exception werfen können, nach der folgenden Regel benannt.

$$throwIf + (Bedingung)$$

Abbildung 6.7: Regel, nach der Methoden für das Error-Handling benannt werden.

Durch das Erstellen von eigenen Exceptions können Szenarien besonders gut beschrieben werden. Beispielsweise das Aktualisieren eines TutorialStates auf den nächsten Schritt, wenn dieser bereits am letzten Schrittes des Tutorials ist, konnte durch die neue `TutorialStateAlreadyAtLastStepException` beschrieben und eindeutig identifizierbar gemacht werden.

6.1.4 Dateninitialisierung

Die Backendimplementierung beinhaltet die Initialisierung von Daten, die vom System vordefiniert werden. Dies betrifft zuerst die `Steps`, da diese nur vom System erstellt werden sollen. Auch gibt es für die verschiedenen Bereiche von QDAcity Tutorials, welche dem Nutzer diese erklären sollen. Dies wird in der `TutorialInitialization` Klasse erledigt und von einem Java Servlet beim Start des Servers ausgeführt, sodass diese Daten automatisch sofort zur Verfügung stehen. Es werden dabei die folgenden Aufgaben übernommen:

- Erstellen von fehlenden Schritten
- Löschen von Schritten, die nicht mehr als *UserAction* vorhanden sind
- Erstellen von fehlenden Basis Tutorials
- Löschen von veralteten Basis Tutorials und allen dazu gehörigen *Tutorial-States* und neues Erstellen dieser Basis Tutorials

Dadurch, dass Tutorials und Schritte nicht nur einmalig in die Datenbank eingefügt, sondern ständig aktuell gehalten werden, können ohne Mehraufwand neue Schritte und Tutorials hinzugefügt oder bestehende Daten angepasst werden. Dies ist besonders im weiteren Verlauf der Entwicklung des Tutorial Systems wichtig, da so die Daten in der Produktivumgebung nicht in einen ungültigen Zustand gelangen.

6.2 Frontend

In diesem Kapitel wird die Implementierung des Tutorial Systems im JavaScript Client beschrieben, die das Verwalten der Daten, das Erkennen von bestimmten Schritten und Erweiterungen der Benutzeroberfläche beinhaltet. Dabei wurde zunächst geprüft, ob und wie die in Kapitel 5.1.3 vorgestellten Designpattern mit React implementiert werden können.

6.2.1 React Context

React bietet dabei selbst einige Mechanismen an, die zur Implementierung verwendet werden können und nicht erst mit JavaScript umgesetzt werden müssen. So kann die React *Context API*[1] genutzt werden, um die Eigenschaften der beiden vorgestellten Designpattern zu erhalten. In React reichen Komponenten Daten typischerweise von oben nach unten über *props* weiter. Dies ist jedoch besonders aufwendig, wenn mehrere einzelne Komponenten die gleichen Daten brauchen und eine Vielzahl von Komponenten zwischen diesen liegen, was auch als *Prop Drilling*[2] bezeichnet wird. Hierbei hilft der *Context*, weil die Daten innerhalb von diesem nur einmal geladen werden müssen und dann von den Komponenten direkt genutzt werden können. Der `TutorialContext` definiert hierbei eine Art Interface mit allen Daten und Funktionen und im `TutorialProvider` befindet sich die eigentliche Implementierung dieser Methoden und Verwaltung der Daten.

Damit der `TutorialContext` für Komponenten bereitsteht, müssen diese im Komponentenbaum von React in den `TutorialProvider` gehüllt werden.

[1] https://reactjs.org/docs/context.html
[2] https://kentcdodds.com/blog/prop-drilling

```
<App>
    <TutorialProvider>
        ...
        <ComponentWithAccessToContext/>
        ...
    </TutorialProvider>
    <ComponentWithoutAccessToContext/>
</App>
```

Codebeispiel 4: Verwendung des TutorialProvider im React Komponentenbaum.

Da der TutorialContext nur einmal existiert, können hier die Eigenschaften des Singleton Patterns genutzt werden, um sicherzustellen, dass die Daten in nur einer Komponente konsistent verwaltet werden. Der TutorialProvider lädt beim Start alle benötigten Daten aus dem Backend, sodass diese über den Context verfügbar sind. Neben den Daten sind im TutorialProvider alle Methoden zum Erstellen, Ändern und Löschen von Daten implementiert und die Methoden sind auch durch den Context für die Komponenten verfügbar. Falls sich die Daten ändern, sorgt die Nutzung des Context zudem dafür, dass sich die UI Komponenten neu rendern.

Besonders wichtig ist die Implementierung und Bereitstellung der Methode `triggerStep(UserAction)`, denn durch diese wird, wie im Observer Pattern in Kapitel 5.1.3 beschrieben, ermöglicht, dass die Komponenten den TutorialContext benachrichtigen, wenn eine bestimmte Handlung durchgeführt wurde. In Abbildung 6.8 wird gezeigt, dass alle UI Komponenten den TutorialProvider über einen Schritt informieren können und sollte der Nutzer gerade ein Tutorial durchführen und der Schritt mit dem aktuell auszuführenden Schritt übereinstimmen, wird der TutorialState im Backend und die vom TutorialProvider verwalteten Daten aktualisiert. Die aktualisierten Daten, wie der neue durchzuführende Schritt, werden dann wieder an die UI Komponenten gereicht und diese rendern sich automatisch neu, wenn die aktualisierten Daten verwendet werden.

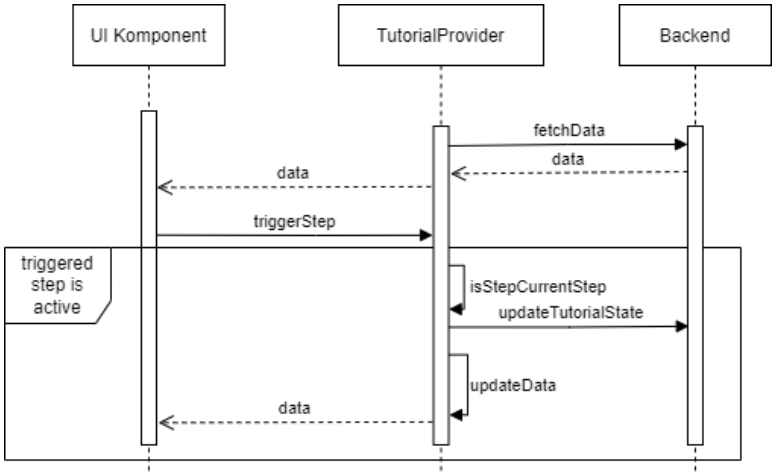

Abbildung 6.8: Sequenzdiagramm zur Darstellung, wie der TutorialProvider über Schritte banachrichtigt wird.

6.2.2 Übersicht der Tutorials

Nachdem durch den `TutorialContext` die benötigten Daten geladen werden, können dem Nutzer auf der Übersichtsseite der Tutorials durch die *TutorialOverview* Komponente alle für ihn verfügbaren Tutorials angezeigt werden. Diese setzen sich aus dem aktuell aktiven Tutorial, den Basis Tutorials, den vom Nutzer selbst erstellten Tutorials und den von anderen Nutzern gestarteten Tutorials zusammen.

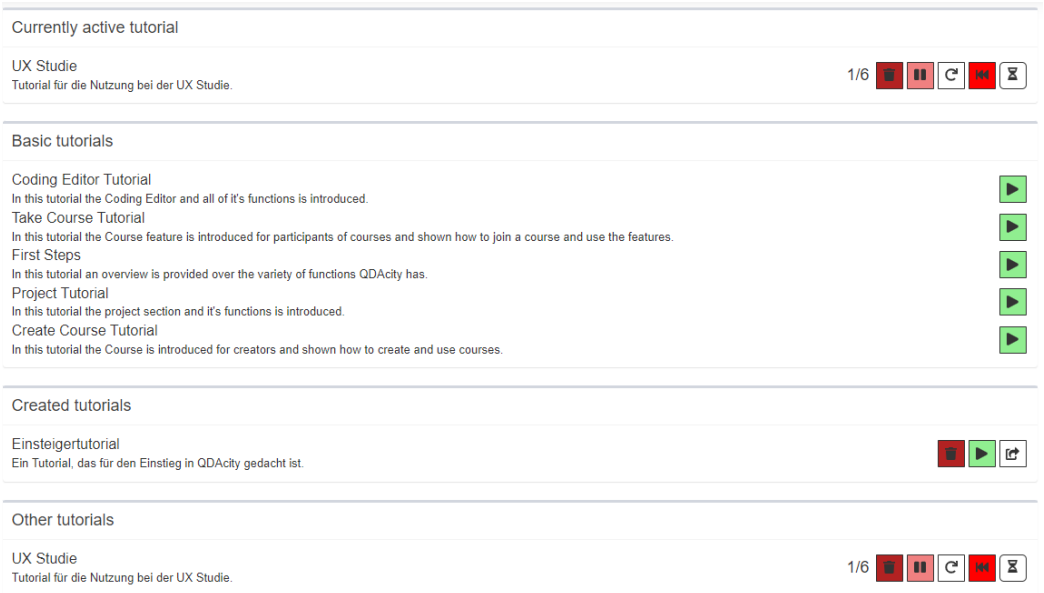

Abbildung 6.9: UI für die Übersicht aller verfügbaren Tutorials eines Nutzers.

Die `TutorialOverview` Komponente setzt sich aus vier einzelnen Komponenten zusammen:

- CurrentlyActiveTutorial: Zeigt das gerade aktive Tutorial an
- BasicTutorialsOverview: Zeigt alle Basis Tutorials an
- UserTutorialsOverview: Zeigt alle vom Nutzer erstellen Tutorials an
- ImportedTutorialsOverview: Zeigt alle fremden Tutorials an

Da diese Komponenten alle Tutorials rendern, wurde mit dem `TutorialPanel` eine wiederverwendbare Komponente geschaffen, die ein einzelnes Tutorial darstellt und verschiedene Buttons je nach Daten des Tutorials über die Komponente `TutorialButtons` anzeigt. Die Komponente `TutorialList` nutzt das TutorialPanel, um gleich eine ganze Liste an Tutorials darstellen zu können, was BasicTutorialsOverview, UserTutorialsOverview und ImportedTutorialsOverview benötigen.

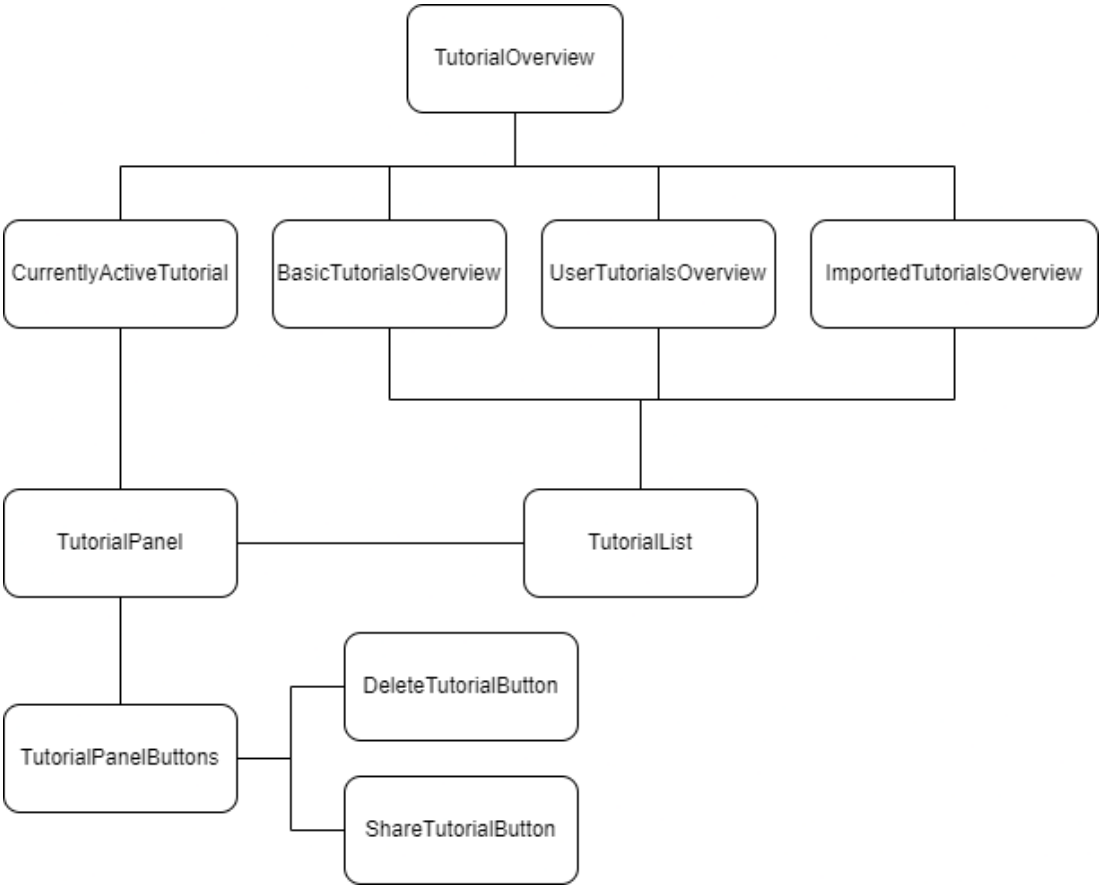

Abbildung 6.10: UI Komponenten, die für die Übersicht aller verfügbaren Tutorials eines Nutzers verwendet werden.

Neben der Anzeige der Tutorials stehen durch die Buttons auch eine Vielzahl an Funktionen für die Tutorials zur Verfügung.

- Starten eines Tutorials
- Stoppen eines Tutorials, falls dieses aktiv ist
- Zurücksetzten des Fortschrittes, falls einer vorhanden ist
- Löschen des Fortschrittes, falls einer vorhanden ist
- Löschen des ganzen Tutorials, falls der Nutzer der Ersteller ist
- Teilen eines Tutorials, falls der Nutzer der Ersteller ist

6.2.3 Tutorial Durchführung

Wenn ein Tutorial aus der Tutorialübersicht gestartet wurde, wird der Nutzer auf das *PersonalDashboard* von QDAcity geleitet und seine Benutzeroberfläche für das Tutorial verändert. So wird bei jedem Schritt die Komponente hervorgehoben (gehighlighted), die der Nutzer entweder anklicken soll oder die für die Ausführung des Schrittes wichtig ist. Der Rest der Benutzeroberfläche wird durch ein verschwommenes Overlay überdeckt, damit der Nutzer sich nur auf bestimmte Elemente fokussiert. Zudem gibt es eine Sidebar, in der dem Nutzer Informationen über das Tutorial, den aktuellen Schritt und seinen Fortschritt angezeigt werden.

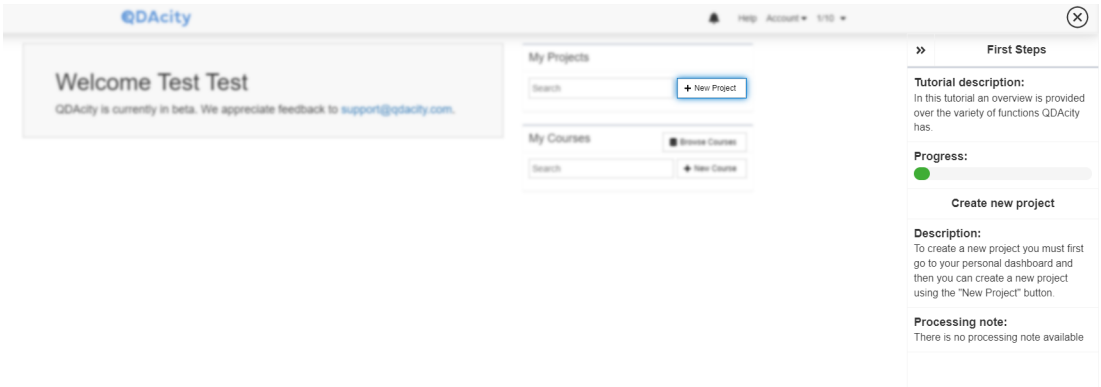

Abbildung 6.11: Die Benutzeroberfläche beim Durchführen eines Tutorials.

Highlighting

Das Hervorheben von Elementen soll durch einen blau schimmernde Umrandung erfolgen. Dabei wurde für die Implementierung eine Lösung gewählt, die bei allen hervorzuhebenden Elementen wiederverwendet werden kann. Dies ist nötig, da sonst ein enormer Aufwand entstanden wäre, weil für jeden neuen Schritt das

Highlighting separat implementiert oder angewendet werden müsste. Durch die `HighlightableComponent` existiert lediglich eine Implementierung in Form einer React Komponente und die Elemente müssen im React Komponentenbaum nur in diese gehüllt werden. Dabei werden die HTML Elemente in ein `<div>` gehüllt, welches die benötigte blau schimmernde Umrandung besitzt. Zudem wird in der `HighlightableComponent` noch das verschwommene Overlay und ein Button gerendert, der das Highlighting wieder entfernt.

```
return (
    <div>
        <HighlightOverlay isVisible={this.isOverlayVisible()} />
        <ExitButton
            title={exitButtonTooltip}
            onClick={this.closeHighlighting}
            isVisible={this.isOverlayVisible()}
        >
            <FontAwesomeIcon icon={faTimesCircle} size="fa-lg" />
        </ExitButton>
        <HighlightedComponent
            padding={this.props.padding}
            borderRadius={this.props.borderRadius}
        >
            {this.props.children}
        </HighlightedComponent>
        <TutorialExplanationStep
            userAction={this.props.userAction}
        />
    </div>
);
```

Codebeispiel 5: HTML Code, der von der HighlightableComponent gerendert wird.

Die Elemente sollen dabei nur hervorgehoben werden, wenn ein Tutorial aktiv ist und der aktuelle Schritt diese Elemente benötigt, ansonsten sollen die Elemente nicht verändert werden. Dies wird durch den `ActiveTutorialStepWrapper` ermöglicht, der anstelle der `HighlightableComponent` um die Elemente gehüllt wird. Der `ActiveTutorialStepWrapper` überprüft, ob ein angegebener Schritt aktiv ist und falls dies der Fall ist, wird das Element mit *Highlighting* gerendert. Dabei können durch *props* verschiedene Eigenschaften angepasst werden, wie das Deaktivieren des Overlays, des Blockierens von Events oder optische Eigenschaften des *Highlightings*.

```
<ActiveTutorialStepWrapper
   userActions={[UserAction.ENABLE_UML_EDITOR]}
   disableHighlighting={false}
   disableBlur={true}
   disableEventBlocking={true}
   padding={2}
   borderRadius={5}
>
   <Element/>
</ActiveTutorialStepWrapper>
```

Codebeispiel 6: Verwendung des ActiveTutorialStepWrapper im React Komponentenbaum.

Obwohl das *Highlighting* nur einmal implementiert wurde, müssen für alle 51 Schritte die Elemente im React Komponentenbaum gefunden werden, die hervorgehoben werden sollen und diese mit dem `ActiveTutorialStepWrapper` umhüllt werden. Dabei mussten teilweise Änderungen an den bestehenden CSS Eigenschaften vorgenommen werden, damit durch das Hinzukommen der `HighlightableComponent` keine ungewollte Effekte im Layout entstehen.

Zudem gibt es neben der standardmäßigen Hervorhebung von Elementen noch spezielle Schritte, welche dem Nutzer nur Informationen anzeigen sollen und keine direkte Handlung von diesem benötigen. Dafür wurde in der *HighlightableComponent* noch der `TutorialExplanationStep` gerendert, der nur dann etwas anzeigt, wenn für diesen Schritt Inhalte in der `TutorialStepsMessages` Klasse hinterlegt sind. Diese hält für alle Schritte den Titel und die Beschreibung in der passenden Sprache und für manche Schritte auch den Inhalt für den `TutorialExplanationStep` bereit.

So ist die Verwendung von Codes im CodingEditor von QDAcity keine besonders intuitive Aufgabe, welche auch mit Hervorhebung der einzelnen Elemente dem Nutzer nur schwer vermittelt werden kann. Deshalb wurde ein Schritt mit einem `TutorialExplanationStep` erstellt, der ein Fenster rendert, in dem ein GIF angezeigt wird, dass dem Nutzer genau zeigt, wie Codes verwendet werden. Der Nutzer kann so nachvollziehen, wie Codes genutzt werden und anschließend über einen Button zum nächsten Schritt des Tutorials gelangen.

Abbildung 6.12: Erklärung der Verwendung von Codes in QDAcity durch einen TutorialExplanationStep.

Aktualisierung des Schrittes

Sobald der Nutzer den hervorgehobenen Schritt ausgeführt hat, muss der `TutorialContext` benachrichtigt werden, um den TutorialState im Backend zu aktualisieren, den nächsten Schritt zu laden und dem Nutzer anzeigen und hervorheben zu können. Wie bereits in Kapitel 6.2.1 beschrieben, wird dafür auch der TutorialContext genutzt. Für alle 51 Schritte wurden die Codestellen gefunden, an welchen der Nutzer den spezifischen Schritt ausgeführt hat und über den Aufruf der `triggerStep` Methode der `TutorialContext` informiert werden muss.

6.2.4 Create new Tutorial

Neben den Basis Tutorials können Nutzer auch selbst Tutorials erstellen. Die `CreateTutorialPage` Komponente stellt dafür die Benutzeroberfläche aus den folgenden Bereichen bereit:

- Eingabe des Tutorial Namen
- Hinzufügen einer Beschreibung des Tutorials in einer der verfügbaren Sprachen
- Liste aller verfügbaren Schritte
- Liste aller zum Tutorial hinzugefügten Schritte, mit der Möglichkeit Bearbeitungsvermerke in den verfügbaren Sprachen hinzuzufügen

Um ein Tutorial erstellen zu können, muss der Nutzer einen gültigen Namen eingeben, eine Beschreibung hinzufügen und mindestens einen Schritt hinzufügen, wobei Schritte auch mehrfach hinzugefügt werden können. Dies wird validiert

und nur nach einer erfolgreichen Validierung wird das Tutorial erstellt. Bei einer fehlerhaften Validierung wird dem Nutzer angezeigt, welche Fehler aufgetreten sind.

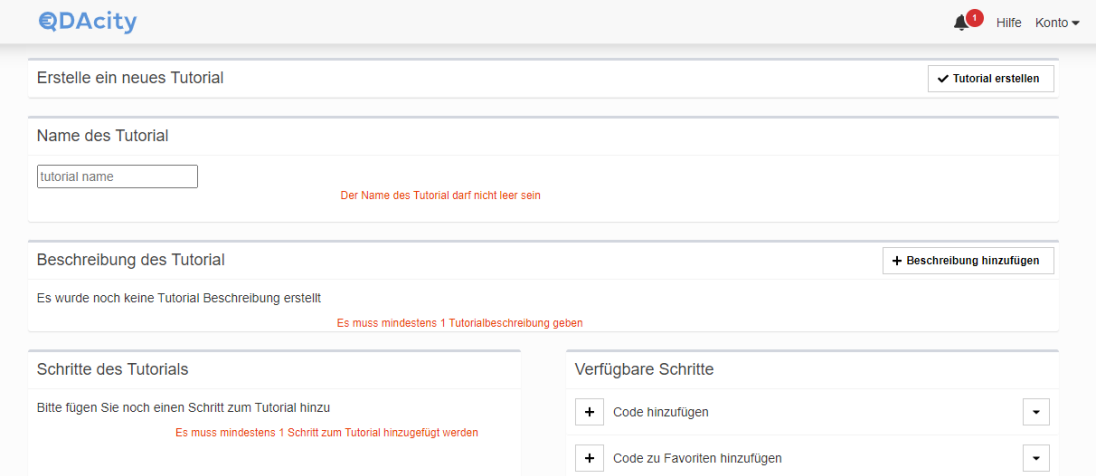

Abbildung 6.13: Anzeige der roten Fehlermeldungen bei einer erfolglosen Validierung.

Der Nutzer kann für die einzelnen Schritte Bearbeitungsvermerke hinzufügen, dies ist jedoch nur optional. Das Eingabefenster für die Erstellung von Bearbeitungsvermerken und Beschreibungen des Tutorials ist dabei dieselbe Komponente in Form des `CreateInternationalizedTextModal`, da dieses wiederverwendet werden kann. Das `CreateInternationalizedTextModal` lässt den Nutzer eine Sprache auswählen und einen Text eingeben, der ebenso validiert wird.

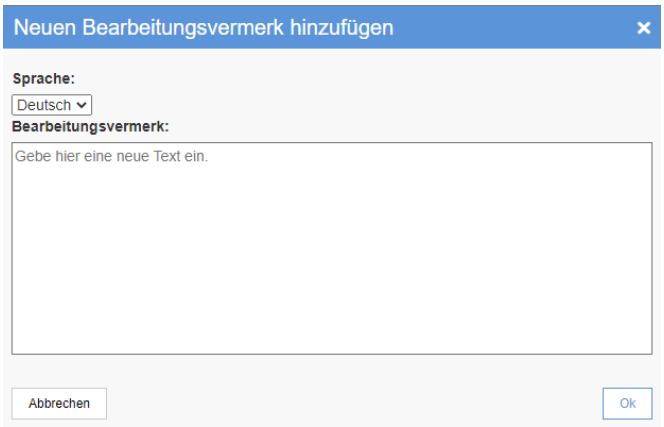

Abbildung 6.14: Fenster zur Erstellung eines Bearbeitungsvermerkes.

Über den Button in der oberen rechten Ecke der Benutzeroberfläche kann das Tutorial erstellt werden, woraufhin es dem Nutzer dann in der Tutorialübersicht angezeigt wird und auch mit anderen Nutzern geteilt werden kann.

6.2.5 Tutorial teilen

Die vom Nutzer erstellen Tutorials können durch einen Link geteilt werden, indem dieser von anderen Nutzern im Webbrowser aufgerufen wird. Der Ersteller eines Tutorials findet in der Tutorialübersicht einen Button, der diesen Link anzeigt und die Möglichkeit bietet, den Link in die Zwischenablage zu kopieren oder den Link neu zu generieren und damit den bestehenden Link zu invalidieren.

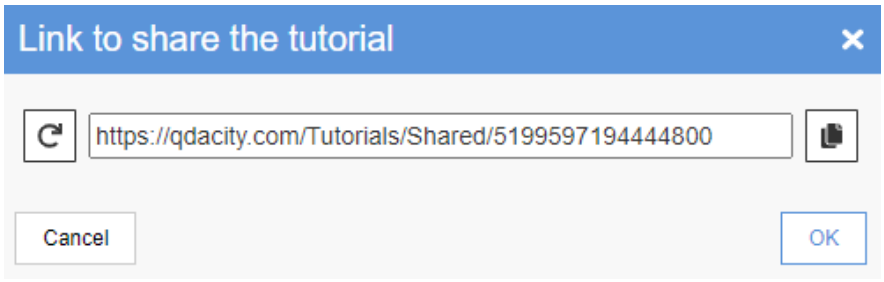

Abbildung 6.15: Anzeige des Links, mit welchem ein bestimmtes Tutorial geteilt werden kann.

Sobald ein anderer Nutzer den Link aufruft, kann dieser das Tutorial über einen Button starten, außer der Nutzer hat dieses Tutorial bereits gestartet oder der Nutzer hat bereits ein aktives Tutorial. Wenn der Nutzer das Tutorial nicht starten kann wird dieser auf die Tutorialübersicht weitergeleitet und gegebenfalls eine Fehlermeldung angezeit.

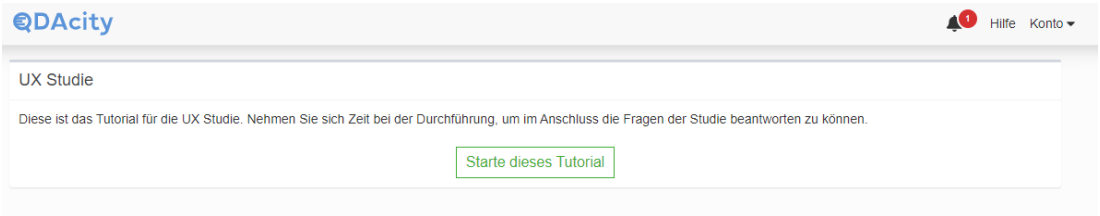

Abbildung 6.16: Benutzeroberfläche, um ein Tutorial eines fremden Nutzers zu starten.

7 Evaluation

In diesem Kapitel wird zum Einen evaluiert, ob die funktionalen und nicht-funktionalen Anforderungen aus Kapitel 3 erfüllt wurden und zum Anderen eine Studie durchgeführt, um das Tutorial System hinsichtlich seiner UX-Eigenschaften zu untersuchen.

7.1 Funktionale Anforderungen

Zunächst werden die FA an das Tutorial System aus Kapitel 7.1 evaluiert.

7.1.1 Vordefinierte Tutorials

FA 1: Das Tutorial System soll Tutorials, die im Vorfeld vom System erstellt wurden und nicht mehr durch den Nutzer erstellt werden müssen, laden und bereitstellen können.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Auf der Übersichtsseite der Tutorials werden die vordefinierten Basis Tutorials geladen und jedem Nutzer angezeigt. Um die Daten zu laden, wurde im Frontend der *GET /tutorials/basicTutorials* REST Endpunkt verwendet, der vom Backend implementiert wurde.

FA 2: Das Tutorial System soll vordefinierte Tutorials für die in Kapitel 2 vorgestellten Bereiche bereitstellen.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

In der Tutorialinitialisierung werden fünf Basis Tutorials erstellt, welche dem Nutzer die Funktionalität von QDAcity näher bringen sollen.

- *First Steps Tutorial:* Einstieg in QDAcity und ein grober Überblick über Projekte und den Coding Editor
- *Project Tutorial:* Tutorial, das die Funktionen der Projektübersicht erklärt

- *Coding Editor Tutorial:* Tutorial, das die Funktionen des Coding Editros erklärt
- *Create Course Tutorial:* Tutorial, das erklärt wie in QDAcity Kurse erstellt werden können
- *Join Course Tutorial:* Tutorial, das erklärt wie Nutzer in QDAcity an Kursen teilnehmen können

Mit diesen Basis Tutorials wurden alle Bereiche von QDAcity, wie sie in Kapitel 2 beschrieben wurden, durch Tutorials abgebildet. Dabei konnte die wichtigste Funktionalität dieser Bereiche in den Tutorial abgebildet werden.

7.1.2 Tutorials

FA 3: Das Tutorial System soll vom Nutzer erstellte Tutorials laden und anzeigen können.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Auf der Übersichtsseite der Tutorials werden die vom Nutzer erstellten Tutorials geladen und angezeigt. Im Frontend wurde, um die Daten zu laden, der *GET /tutorials/userTutorials* REST Endpunkt verwendet, der vom Backend implementiert wurde.

FA 4: Der Nutzer soll Tutorials erstellen können.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Der Nutzer kann durch die von der `CreateTutorialPage` Komponente bereitgestellte Funktionalität, Tutorials selbst erstellen. Der Nutzer kann die Schritte des Tutorials selbst auswählen, dem Tutorial einen Namen geben und Beschreibungen und Bearbeitungsvermerke hinzufügen. Neben der `CreateTutorialPage` Komponente werden dafür auch der *POST /tutorials* REST Endpunkt benötigt, der im Backend implementiert wurde und die Daten persistent in der Datenbank abspeichert.

FA 5: Der Nutzer soll den Namen, die Beschreibung oder die Schritte von selbst erstellten Tutorials ändern können.

✗ Die Funktionale Anforderung wurde teilweise erfüllt.

Nutzer sollten die von ihnen erstellten Tutorials ändern können, dafür wurde jedoch noch keine Benutzeroberfläche implementiert. Im Backend wurde mit dem REST Endpunkt *PUT /tutorials/tutorialId* bereits die Grundlage geschaffen, existierende Tutorials zu ändern. Ohne die grafische Benutzeroberfläche ist die Funktionale Anforderung jedoch nicht erfüllt und muss zu einem späteren Zeitpunkt noch implementiert werden, dabei können die Unterkomponenten der zum

Erstellen verwendeten `CreateTutorialPage` Komponente wiederverwendet werden.

FA 6: Der Nutzer soll selbst erstellte Tutorials löschen können.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Die auf der Tutorialübersichtseite angezeigten Tutorials des Nutzers verfügen über einen Button zum Löschen des gesamten Tutorials, wobei der Nutzer darauf hingewiesen wird, dass damit auch alle Fortschritte von anderen Nutzern zu diesem Tutorial gelöscht werden.

Abbildung 7.1: Benutzeroberfläche für ein selbst erstelltes Tutorial mit dem Button, um dieses zu löschen.

FA 7: Der Nutzer soll selbst erstellte Tutorials über eine URL mit anderen Nutzern teilen können.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Wie in Abbildung 7.1 zu sehen ist, verfügt die Benutzeroberfläche von selbst erstellten Tutorials über einen Button zum Teilen des Tutorials, beziehungsweise durch den Button wird dem Nutzer eine URL angezeigt.

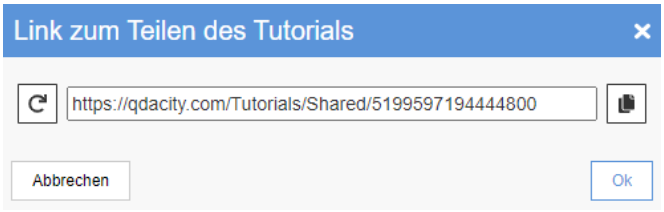

Abbildung 7.2: Benutzeroberfläche zur Anzeige der URL, die zum Teilen des Tutorials benötigt wird.

Der Nutzer kann die URL mit anderen Nutzern teilen und diese können damit das Tutorial starten.

7.1.3 Tutorial Fortschritte

FA 8: Das Tutorial System soll dem Nutzer zu jedem Tutorial seinen Fortschritt anzeigen, falls vorhanden. Dabei soll der Status, die Anzahl der abgeschlossenen und noch zu absolvierenden Schritte angezeigt werden.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Auf der Tutorialübersichtseite wird dem Nutzer direkt bei den geladenen Tutorials sein Fortschritt angezeigt, sofern ein Fortschritt vorhanden ist. In Abbildung 7.3 ist zu sehen, wie je nach `TutorialStatus` ein anderes Symbol für aktive, pausierte oder abgeschlossene Tutorials abgebildet und dem Nutzer angezeigt wird, bei welchem Schritt dieser sich gerade im Tutorial befindet. Zudem wird dem Nutzer immer sein gerade aktives Tutorial angezeigt.

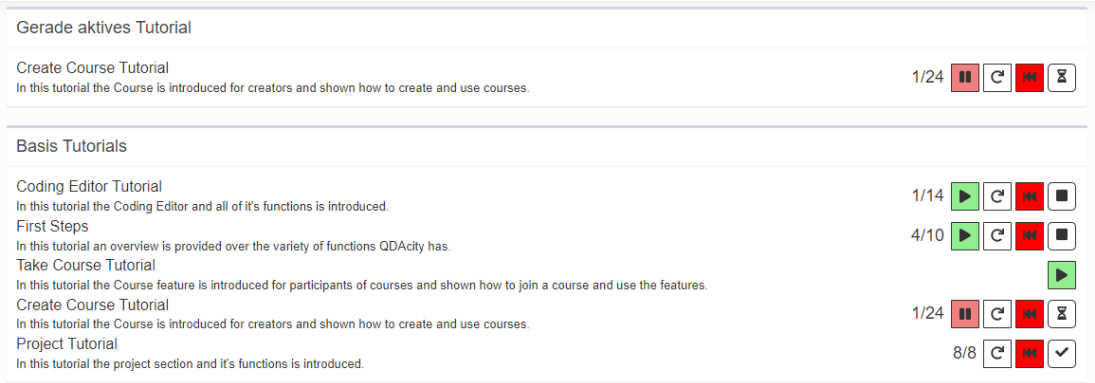

Abbildung 7.3: Abbildung der Fortschritte zu den einzelnen Tutorials.

Neben der Benutzeroberfläche im Frontend wurden auch REST Endpunkte implementiert. Der Endpunkt *GET /tutorials/states*, der alle Fortschritte des Nutzers aus der Datenbank lädt, und der Endpunkt *GET /tutorials/state/current*, der den gerade aktive Tutorial Fortschritt eines Nutzers zurück liefert.

FA 9: Das Tutorial System soll erkennen, wenn einzelne Schritte vom Nutzer ausgeführt werden und den Fortschritt des Nutzers aktualisieren, wenn der durchgeführte Schritt dem gerade aktiven Schritt des Tutorials entspricht.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Durch die Implementierung des `TutorialContext` im Frontend und der Verwendung der `triggerStep()` Methode wie in Kapitel 6.2.3 beschrieben, erkennt das Tutorial System im Frontend sobald ein definierter Schritt durchgeführt wurde und benachrichtigt den `TutorialContext`. Sollte der durchgeführte Schritt der gerade aktive Schritt eines Tutorials sein, wird dieser daraufhin auch im Backend durch die REST Endpunkte *POST /tutorials/states/tutorialStateId/nextStep* und *PUT /tutorials/states/tutorialStateId* aktualisiert und der nächste Schritt des Tutorials angezeigt. Die `triggerStep()` Aufrufe wurden für alle verfügbaren Schritte implementiert.

FA 10: Das Tutorial System soll dem Nutzer anzeigen, welcher Schritt des Tutorials durchzuführen ist.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

In der Tutorial Sidebar wird der aktive Tutorialfortschritt und der aktuell durchzuführende Schritt angezeigt. Dabei werden dem Nutzer bei der Durchführung des Tutorials neben dem Namen und der Beschreibung des Tutorials, auch der Name, die Beschreibung und falls vorhanden der Bearbeitungsvermerk des gerade aktive Schrittes angezeigt. In einer Fortschrittsanzeige kann der Nutzer erkennen, wie weit dieser im Tutorial bereits vorangekommen ist. Die `TutorialSidebar` Komponente wird bei der Durchführung des Tutorials aktualisiert, sobald ein Schritt ausgeführt wurde.

FA 11: Das Tutorial System soll die HTML Elemente optisch hervorheben, die der Nutzer zur Durchführung des gerade aktiven Schrittes benötigt.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Durch die Implementierung des `ActiveTutorialStepWrapper` und des `HighlightableComponent` können Elemente für bestimmte Schritte eines Tutorials hervorgehoben und dem Nutzer so gezeigt werden, welche dieser zur Durchführung des Schrittes benötigt. Dabei kann durch die Verwendung eines verschwommenen Overlays die restliche Benutzeroberfläche in den Hintergrund gerückt werden, sodass der Nutzer nur die hervorgehobenen Elemente nutzen kann. Dabei wurden für alle 51 Schritte die Elemente identifiziert und in den `ActiveTutorialStepWrapper` gehüllt, wie in Kapitel 6.2.3 beschrieben.

FA 12: Das Tutorial System soll dem Nutzer anzeigen, wenn ein Schritt erfolgreich ausgeführt wurde.

✗ Die Funktionale Anforderung wurde nicht erfüllt.

Um dem Nutzer zu zeigen, dass ein Schritt erfolgreich ausgeführt wurde und nun ein neuer Schritt durchzuführen ist, sollte eine Animation in der Tutorial Sidebar eingefügt werden. Diese Animation sollte als Übergang zwischen den Schritten dienen und den Nutzer deutlich auf die Veränderung hinweisen, was jedoch im Rahmen dieser Arbeit nicht mehr implementiert werden konnte.

Bei der erfolgreichen Ausführung ändern sich in der Sidebar der Name, die Beschreibung und falls vorhanden der Bearbeitungshinweis.

FA 13: Das Tutorial System soll dem Nutzer anzeigen, wenn das Tutorial erfolgreich abgeschlossen wurde.

✗ Die Funktionale Anforderung wurde nicht erfüllt.

Nach Vollendung des letzten Schrittes sollte dem Nutzer angezeigt werden, dass dieser das Tutorial abgeschlossen hat. Dies wurde im Rahmen dieser Arbeit noch nicht implementiert. Nach Durchführung des letzten Schrittes verschwindet nur die Sidebar und alle Hervorhebungen.

Dem Nutzer sollte dabei eigentlich in einem Fenster angezeigt werden, dass das

Tutorial abgeschlossen ist und weitere Tutorials vorgeschlagen werden, die der Nutzer noch nicht absolviert hat und dann starten könnte.

FA 14: Der Nutzer soll Tutorials starten können.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Durch den Button zum Starten eines Tutorial in der Tutorialübersicht oder durch das Aufrufen einer URL, mit der ein Tutorial geteilt wurde, kann ein Tutorial gestartet werden. Neben den Komponenten im Frontend kann durch die REST Endpunkte *POST /tutorials/tutorialId/state* und *POST /tutorials/state/tutorialLinkId* ein *TutorialState* für das Tutorial erstellt und in der Datenbank gespeichert werden.

FA 15: Der Nutzer soll aktive Tutorials stoppen können.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Ein aktives Tutorial kann durch den Button zum Stoppen in der Tutorialübersicht pausiert werden. Der Button ruft dabei den REST Endpunkt *PUT /tutorials/states/tutorialStateId* auf, der den TutorialStatus auf *INACTIVE* setzt.

FA 16: Der Nutzer soll pausierte Tutorials weiterführen können.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Ein pausiertes Tutorial kann durch den Button zum Weiterführen in der Tutorialübersicht aktiviert werden. Der Button ruft dabei den REST Endpunkt *PUT /tutorials/states/tutorialStateId* auf, der den TutorialStatus auf *ACTIVE* setzt. Sollte bereits ein Tutorial aktiv sein, wird der Nutzer durch eine Fehlermeldung informiert.

FA 17: Der Nutzer soll seinen Tutorialfortschritt löschen können.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Sollte ein Fortschritt für ein Tutorial existieren, kann dieser durch den Button zum Löschen in der Tutorialübersicht aus der Datenbank entfernt werden. Dabei wird der REST Endpunkt *DELETE /tutorials/states/tutorialStateId* aufgerufen, der die Daten aus der Datenbank löscht und die gecachten Daten invalidiert.

FA 18: Der Nutzer soll seinen Tutorialfortschritt zurücksetzen können.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Der Fortschritt für ein Tutorial kann durch den Button zum Zurücksetzen auf den ersten Schritt des Tutorials gesetzt werden. Dabei wird der REST Endpunkt *POST /tutorials/states/tutorialStateId/resetState* aufgerufen.

FA 19: Das Tutorial System soll Fortschritte von Tutorials korrekt zurücksetzen, wenn diese geändert werden.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Obwohl Nutzer derzeit noch keine Benutzeroberfläche zum Ändern von Tutorials haben, existiert im Backend bereits der REST Endpunkt und die Businesslogik dafür. Bei Anpassungen am Tutorial sind besonders Veränderungen bei den Schritten eines Tutorials kritisch. Sollte ein oder mehrere Schritte entfernt, hinzugefügt oder in der Reihenfolge getauscht werden, muss überprüft werden, ob bestehende Fortschritte des Tutorials auf den ersten Schritt zurückgesetzt werden müssen.

Wie die Formel 7.4 zeigt, werden *TutorialStates* immer dann zurückgesetzt, wenn die Position des Schrittes, den der Nutzer aktuell ausführen soll, größer oder gleich der niedrigsten Position der geänderten Schritte ist.

$$Position_{TutorialState} \geq Position_{Änderung}$$

Abbildung 7.4: Formel, nach der Fortschritte von Tutorials bei Änderungen des Tutorials zurückgesetzt werden.

FA 20: Das Tutorial System soll Fortschritte löschen, wenn das dazugehörige Tutorial gelöscht wird.

✓ Die Funktionale Anforderung wurde vollständig erfüllt.

Sollte ein Tutorial gelöscht werden, werden auch alle Fortschritte von anderen Nutzern gelöscht.

7.2 Nicht-Funktionale Anforderungen

Im Folgenden werden die NFA aus Kapitel 4.2 evaluiert.

7.2.1 Funktionale Eignung

NFA 1: Das Tutorial System soll vollständig funktionstüchtig sein und der Nutzer soll Tutorials nutzen können.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Durch Unit-Test, Integrationstests, E2E Tests, manuelles Testen und Code Reviews konnte die volle Funktionalität des Tutorial Systems getestet und überprüft und keine Fehler gefunden werden. Zudem konnten neue Nutzer in der UX Studie über eine URL ein Tutorial starten und durchführen.

NFA 2: Das Tutorial System soll keine existierende Funktionalität beeinträchtigen.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Durch Unit-Tests, Integrationstests, E2E Tests und Code Reviews wurde bei der Entwicklung und Integration des Tutorial Systems stetig geprüft, ob bestehende Funktionalität beeinträchtigt wird. Es konnten dabei keine Beeinträchtigungen festgestellt werden.

7.2.2 Effizienz

NFA 3: Das Tutorial System soll den existierenden Cache nutzen, um die bei Anfragen aus der Datenbank geladene Daten zu cachen.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

In den entwickelten REST Endpunkten wurde, soweit möglich, der Memcache verwendet und geprüft, ob die Daten bereits gecacht sind und anderenfalls die aus der Datenbank geladenen Daten gecacht. Alle Endpunkte, die einzelne Datensätze anhand einer ID laden, nutzen die Schnittstelle des Memcache. Zudem war es wichtig die Basis Tutorials und die Steps zu cachen, da sich diese Daten in der Regel nicht mehr ändern und von allen Nutzern geladen werden, somit ist der Cache besonders effektiv bei diesen Datensätzen. Für beide Datensätze wurde jeweils ein eigener Key angelegt und dieser im Memcache gespeichert, um so alle Schritte oder alle Basis Tutorials aus dem Memcache laden zu können.

7.2.3 Kompatibilität

NFA 4: Das Tutorial System soll mit den in QDAcity verwendeten Google Diensten (Google App Engine, Google Datastore und Google Cloud Endpoints) kompatibel sein.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Bei der Implementierung des Tutorial Systems wurden im Backend dieselben Google Dienste verwendet, die auch im restlichen QDAcity Backend genutzt wurden. So wurde Google Cloud Endpoints zur Erstellung der REST Endpunkte genutzt und der Google Datastore zu Anbindung der Datenbank. Das Backend des Tutorial Systems fügt sich damit nahtlos in das restliche Backend ein und kann auch mit Google App Engine betrieben werden.

NFA 5: Das Tutorial System und die zur Implementierung verwendeten Abhängigkeiten sollen mit den in QDAcity verwendeten Abhängigkeiten kompatibel sein.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Für alle neu in das Projekt hinzugefügten Abhängigkeiten wurden geprüft, ob diese mit den existierenden Abhängigkeiten in QDAcity kompatibel sind und erst

dann wurden diese hinzugefügt. So wurde für die Abhängigkeit `react-spring` nicht die neueste Version genutzt, sondern nur die Version 9.1, da diese noch mit der in QDAcity verwendeten React Version kompatibel ist.

7.2.4 Benutzbarkeit

NFA 6: Die Texte des Tutorial Systems sollen entsprechend der Internationalisierungsstrategie von QDAcity übersetzt werden.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Alle vordefinierten Texte und Beschriftungen wurden mit der Internationalisierungsstrategie von QDAcity umgesetzt und sind somit in Deutsch und in Englisch verfügbar. Zudem wurde die Möglichkeit geschaffen, Inhalte, die der Nutzer erstellt, in den von QDAcity unterstützen Sprachen zu erstellen und zur Verfügung zu stellen. Zur Überprüfung wurde ein Job der CI/CD Pipeline genutzt, der verifiziert, dass für alle Text eine Übersetzung vorhanden ist und alle Übersetzungen verwendet werden.

NFA 7: Das Farbschema des Tutorial Systems soll mit dem von QDAcity übereinstimmen.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Bei der Implementierung der Benutzeroberfläche des Tutorial Systems wurden viele bereits bestehende Komponenten, wie Buttons, Modals oder Prompts, wiederverwendet. Zudem wurden die Farben entweder aus der `Theme.js` verwendet oder dort neu definiert. Beim Hervorheben der Elemente wurde deshalb auch eine blau schimmernde Umrandung gewählt.

7.2.5 Zuverlässigkeit

NFA 8: Das Tutorial System soll HTTP Statuscodes und in der Benutzeroberfläche Fehlermeldungen zurückgeben.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Die REST Endpunkte des Tutorial Systems geben im Backend HTTP Codes[1] zurück, die angeben, ob eine Anfrage erfolgreich war oder ob ein bestimmter Fehler vorliegt. Diese Codes werden im Frontend verwendet, um dem Nutzer Fehlermeldungen anzeigen zu können. So kann beispielsweise erkannt werden, wenn der Nutzer bereits ein aktives Tutorial hat und diesem eine passende Fehlermeldung angezeigt werden.

[1] https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

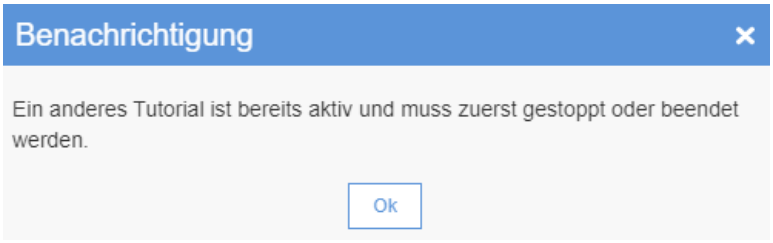

Abbildung 7.5: Fehlermeldung, falls beim Starten eines Tutorials bereits ein aktives Tutorial existiert.

7.2.6 Sicherheit

NFA 9: API Endpunkte sollen nur Anfragen von angemeldeten Nutzern bearbeiten und im Fehlerfall einen HTTP Statuscode zurückliefern.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Die REST Endpunkte des Tutorial Systems können nur von angemeldeten Nutzern aufgerufen werden und im Fehlerfall wird eine `UnauthorizedException` zurückgegeben.

NFA 10: Nutzer dürfen nur ihre eigenen Tutorials ändern oder löschen.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Die REST Endpunkte *DELETE /tutorials/tutorialId* und *PUT /tutorials/tutorialId* überprüfen, welcher Nutzer diese aufruft und geben eine `IllegalAccessException` zurück, wenn der Aufrufer nicht der Ersteller des Tutorials ist.

NFA 11: Nutzer dürfen nur ihre eigenen oder geteilte Tutorials starten, pausieren, weiterführen und zurücksetzten.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Der REST Endpunkt *POST /tutorials/tutorialId/state* zum Starten eines Tutorial überprüft, welcher Nutzer das Tutorial starten möchte und gibt eine `IllegalAccessException` zurück, wenn der Aufrufer nicht der Ersteller des Tutorials ist.

NFA 12: Nutzer dürfen nur auf ihre eigenen Tutorial Fortschritte zugreifen und diese löschen.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Die REST Endpunkte *GET /tutorials/states*, *GET /tutorials/state/current*, *GET /tutorials/state/tutorialStateKey* und *GET /tutorials/tutorialId/state* überprüfen, welcher Nutzer diese aufruft und geben entweder nur die Daten des Aufrufers oder eine `IllegalAccessException` zurück, wenn der Aufrufer nicht der Ersteller des TutorialStates ist.

7.2.7 Wartbarkeit

NFA 13: Für die hinzugefügte Funktionalität soll eine Codeabdeckung von mehr als 90% der *LOC* im Backend durch Unittests und Integrationstests erreicht werden.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Im Backend wurde die Funktionalität sowohl durch Unit- als auch durch Integrationstests überprüft. Dabei wurde besonders auf die Testabdeckung der REST Endpunkte und der Implementierung der Businesslogik geachtet, da sonst nur noch Klassen erstellt wurden, die die Daten repräsentieren und keine Logik implementieren. Insgesamt wurden 243 Tests im Backend erstellt mit einer Testabdeckung von 100% für den TutorialEndpoint und von 95.9% für das Package *com.qdacity.tutorials*.

Klasse	Method, %	Line, %
TutorialEndpoint	100% (25/25)	100% (57/57)
TutorialController	100% (32/32)	97,59% (162 /166)
TutorialStateController	100% (28/28)	100% (154/154)
StepController	100% (5/5)	100% (8/8)
TutorialLinkController	100% (14/14)	95% (57/60)
TutorialInitialization	100% (20/20)	98,59% (140/142)

Tabelle 7.1: Testabdeckung der Businesslogik im Backend.

Dabei wurde besonders darauf geachtet, dass nicht nur Tests für erfolgreiche Aufrufe geschrieben wurden, sondern auch Tests für die Fehlerfälle und um die Grenzen von Parametern abzubilden.

NFA 14: Im Frontend sollen alle vordefinierten Tutorials durch E2E Tests getestet werden.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Für die fünf Basis Tutorials wurde je ein E2E Test erstellt, der das Tutorial startet und jeden Schritt des Tutorials ausführt. Damit kann gewährleistet werden, dass die Basis Tutorials und alle ihre Schritte funktionsfähig sind. Sollte sich beispielsweise im Zuge der Weiterentwicklung von QDAcity die Benutzeroberfläche ändern, können mit Hilfe dieser Test Regression erkannt werden.

7.2.8 Übertragbarkeit

NFA 15: Das Tutorial System soll das Hinzufügen von neuen Schritten unterstützen.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Im Tutorial System können weitere Schritte hinzugefügt werden, um so andere Funktionen abbilden zu können. Dabei müssen im Backend und Frontend die folgenden Punkte hinzugefügt werden:

- Im Backend eine neue `UserAction`
- Im Frontend ein Übersetzung für den Namen und die Beschreibung des Schrittes
- Im Frontend der `triggerStep()` Aufruf zum Erkennen, wann der Schritt durchgeführt wurde
- Im Frontend der `ActiveTutorialStepWrapper`, um das dazugehörige Element hervorzuheben

Dann wird durch die in Kapitel 6.1.4 beschriebene Initialisierung des Tutorial Systems automatisch ein neuer Schritt erstellt, der den Nutzern dann mit Highlighting und Schritterkennung zur Verfügung steht.

NFA 16: Das Tutorial System soll die Webbrowsern Google Chrome und Mozilla Firefox unterstützen.

✓ Die Nicht-Funktionale Anforderung wurde vollständig erfüllt.

Das Tutorial System wurde für die Webbrowsern Google Chrome und Mozilla Firefox entwickelt und in diesen getestet. Um beide Brwoser zu unterstützen wurde deshalb für Mozilla Firefox eine andere Implementierung des Overlays bei der Durchführung des Tutorials verwendet, da Mozilla Firefox die CSS Eigenschaft `backdrop-filter: blur(2px);` nicht unterstützt.

7.3 User Experience Studie

Durch eine UX Studie soll festgestellt werden, ob das Tutorial System Nutzern den gewünschten Effekt des schnellen Kennenlernens der Anwendung bringt und Nutzererfahrungen gesammelt werden, um dieses in Zukunft optimieren zu können.

UX Studien unterscheiden sich dadurch von Usability Studien, dass sich diese nicht nur auf die Aufgaben des Nutzers und mögliche Probleme mit der Anwendung fokussieren, sondern auch auf andere Qualitäten achten wie die entsprechende Optik oder Innovativität der Anwendung. Bei Usability wird besonders auf Objektivität geachtet, wobei es bei UX gerade auf subjektive Wahrnehmungen ankommt (Hassenzahl et al., 2008).

Die Qualitäten bei UX Studien können in pragmatische und hedonische Qualitäten unterteilt werden. Pragmatische Qualitäten beschreiben die Gebrauchstauglichkeit der Anwendung, wohingegen die hedonische Qualität beschreibt in wie

weit die menschlichen Bedürfnisse nach Stimulation und Identität erfüllt werden. Findet der Nutzer die Anwendung modern oder aufregend, spricht man von hedonischer Qualität (Hassenzahl et al., 2003).

7.3.1 Studienarten

Um eine UX Studie zu entwerfen, muss zunächst aus den verschiedenen Arten von Studien, die derzeit in der Industrie und im Akademischen Umfeld genutzt werden, eine passende Studie gewählt werden. Die Studienarten bringen jeweils spezifische Vor- und Nachteile mit sich, die genauer betrachtet werden, um die beste Studienart zur Evaluation des Tutorial Systems zu finden.

Im Folgenden werden die Eigenschaften von Laborstudien, Feldstudien, Umfragen und Expertenevaluationen analysiert und mit den Anforderungen bei der Evaluation des Tutorial Systems verglichen (Roto et al., 2009).

Laborstudien

Bei Laborstudien wird darauf geachtet, dass alle Teilnehmer die Studie unter den gleichen, kontrollierten Bedingungen durchführen, um mögliche Störquellen auszuschließen (Pederson, 2017).

Da es sich bei QDAcity jedoch um eine Webanwendung handelt, die keinen Zugriff über mobile Geräte unterstützt, ist die Umgebung bei allen Nutzern bereits gleich, da diese immer über einen Webbrowser auf die Anwendung zugreifen. Somit haben bereits alle Teilnehmer einen einheitlichen Zugang zur Anwendung. Das Erstellen eines künstlichen Umfelds, das Finden von Teilnehmern und die Durchführung vor Ort machen die Evaluation zudem sehr zeitintensiv und aufwendig.

Feldstudien

Feldstudien finden anders als Laborstudien im natürlichen Umfeld des Studieninhalts statt. Dabei werden Experimente durchgeführt, Beobachtungen gemacht und mit dem Teilnehmer interagiert, was besonders beim Verständnis von Problemen helfen kann(Mısırlı et al., 2014).

Bei der Evaluation des Tutorial Systems werden jedoch keine bestimmten Probleme oder Use Cases untersucht sondern ein besonderer Fokus darauf gelegt, ob und welchen Effekt das Tutorial System im Vergleich zur Nutzung ohne das Tutorial System hat. Ähnlich wie bei Laborstudien sind Feldstudien durch das Finden von Teilnehmern und das Durchführen zeitintensiv und aufwendig.

Umfragen

Bei Umfragen werden Daten gesammelt, indem ein Set an Fragen definiert und anschließend von allen Teilnehmern beantwortet wird. Da Umfragen auch online selbständig durch die Teilnehmer durchgeführt werden können, beschränkt sich der Aufwand auf das Erstellen der Fragen und der Auswertung der Antworten. Umfragen sind daher ideal, um in kurzer Zeit viele neue Nutzer zu erreichen und viele Daten und Erfahrungen zur Anwendung zu erhalten.

Experteninterviews

In Experteninterviews werden wenige UX-Experten ausgewählt und diese führen die Evaluation der Anwendung mit Hilfe ihrer Erfahrung und ihrem Wissen durch. Dies kann zwar den Aufwand verringern, da keine passenden Teilnehmer und nur wenige Experten gefunden werden müssen, jedoch befinden sich die UX Kompetenzen im QDAcity Team erst im Aufbau und können daher noch nicht genutzt werden. Zudem wären diese bereits erfahrene Nutzer der Anwendung und es wäre schwierig festzustellen welchen Nutzen das Tutorial System beim Kennenlernen der Anwendung bei diesen hätte.

7.4 Studiendesign

Zunächst muss eine geeignete Studienart gewählt und ein Design für die UX Studie entwickelt werden.

Zur Evaluation des Tutorial Systems sollen neue Nutzer erreicht und Daten darüber erhoben werden, wie das Tutorial System diesen beim Kennenlernen mit QDAcity hilft. Die Studie soll dabei nicht zeitintensiv oder aufwendig sein. Um diese Kriterien zu erfüllen, wurde als Studienart eine Umfrage gewählt und besonders vorteilhaft ist dabei, dass QDAcity als Webanwendung leicht online von Teilnehmern genutzt werden kann und diese dann auch online die Umfrage ausfüllen können. So kann ein Link zur Umfrage geteilt und leicht neue Nutzer erreicht werden.

Teilnehmer der Studie sollen zuerst ein Tutorial durchführen und danach die Fragen der Umfrage beantworten. Dafür wurde ein Tutorial mit dem Namen *Tutorial UX Survey* und den folgenden Schritten erstellt:

- Erstelle ein neues Projekt
- Öffne das erstellte Projekt
- Bearbeite die Projektbeschreibung
- Erstelle ein Projekt To-Do

- Öffne den Coding Editor
- Füge ein Dokument hinzu
- Füge einen Code hinzu
- Schritt, der die Benutzung von Codes erklärt
- Verwendung des erstellten Codes

Das erstellte Tutorial entspricht dabei dem *First Steps Tutorial*, das dem Nutzer in wenigen Schritten einen Großteil der QDAcity Anwendung zeigen soll. Lediglich der letzte Schritt des *First Steps Tutorials*, in dem die Tutorialübersicht geöffnet werden soll, fehlt im erstellten Tutorial.

Damit die Teilnehmer das erstellte Tutorial durchführen können, wird ein Link zum Teilen generiert. Dieser Link wird in Studie eingefügt und die Teilnehmer können so das Tutorial starten, bevor die Fragen beantwortet werden. Welche Fragen die Teilnehmer in der Studie beantworten sollen, wird im folgenden Kapitel beschrieben.

7.4.1 UX Fragebogen

Da es sich bei der Studie um eine Umfrage handelt, musste ein Fragebogen ausgewählt werden, der geeignet ist die UX Eigenschaften des Tutorial Systems zu evaluieren. Dabei kann auf standardisierte und bereits validierte Fragebögen zurückgegriffen werden. Fragebögen mit einer hohen Anzahl an Fragen können die Nutzererfahrung besonders genau und in verschiedenen Bereichen bestimmen. Dies hat jedoch den Nachteil, dass der Aufwand für die Teilnehmer stark steigt und diese die Fragebögen eventuell nicht komplett ausfüllen. Kürzere Fragebögen können die Nutzererfahrung auch evaluieren, ohne dabei Daten durch unvollständige Antworten zu verlieren. Durch die Wahl des Fragebogens kann zudem ein bestimmter Fokus bei der Evaluation gelegt werden(Laugwitz et al., 2008).

Auswahl

Für die UX Studie wurden verschiedene standardisierte UX Fragebögen verglichen, wie User Experience Questionnaire (UEQ), AttrakDiff2 und meCUE (Thomaschewski et al., 2018).

Für die UX Studie zur Evaluation des Tutorial Systems wurde AttrakDiff2 ausgewählt, da eine kürzere Version davon verfügbar ist und das Erstellen, die Durchführung und die Auswertung der Studie durch die Plattform esurvey (https://esurvey.uid.com/) übernommen werden kann und somit nur ein geringer Aufwand besteht.

Der meCUE Fragebogen ist zwar modular, aber eine geringe Anzahl von Fragen kann nur durch das Weglassen von ganzen Modulen erreicht werden. UEQ verfügt auch über eine kürzere Version mit nur acht Fragen, das Erstellen, die Durchführung und die Auswertung der Studie muss jedoch selbst übernommen werden, da keine Plattform dafür bereitsteht.

AttrakDiff2

Der Fragebogen AttrakDiff2 bietet bei der Durchführung der Studien mehrere Versionen des Fragebogens und der Art der Durchführung an. Die Studien können als Einzelauswertung, Vergleich von zwei Anwendungen oder als Vorher-Nachher-Vergleich durchgeführt werden, was im Falle des Tutorial Systems in Zukunft genutzt werden könnte, um Verbesserungen oder Weiterentwicklungen evaluieren zu können. AttrakDiff2 existiert in einer Version mit 28 Fragen und einer gekürzte Version mit nur zehn Fragen. Die Fragen dienen dazu die hedonischen und pragmatischen Qualitäten der Anwendung zu ermitteln und können mit Werten von -3 bis +3 beantwortet werden. Ziel des Tutorial Systems sollte dabei sein, dass beide Qualitäten stark vorhanden sind. Es wird dabei auch verstärkt auf die hedonischen Qualitäten, die sich in die Bedürfnisse Stimulation und Identität unterteilen lassen, geachtet werden (Hassenzahl et al., 2003).

- Stimulation: Wie aufregend oder interessant ist es die Anwendung zu nutzen?
- Identität: Wie wird die Nutzung der Anwendung von anderen wahrgenommen und passt dies zur gewünschten Identität? Wie modern oder cool wird die Nutzung der Anwendung wahrgenommen?

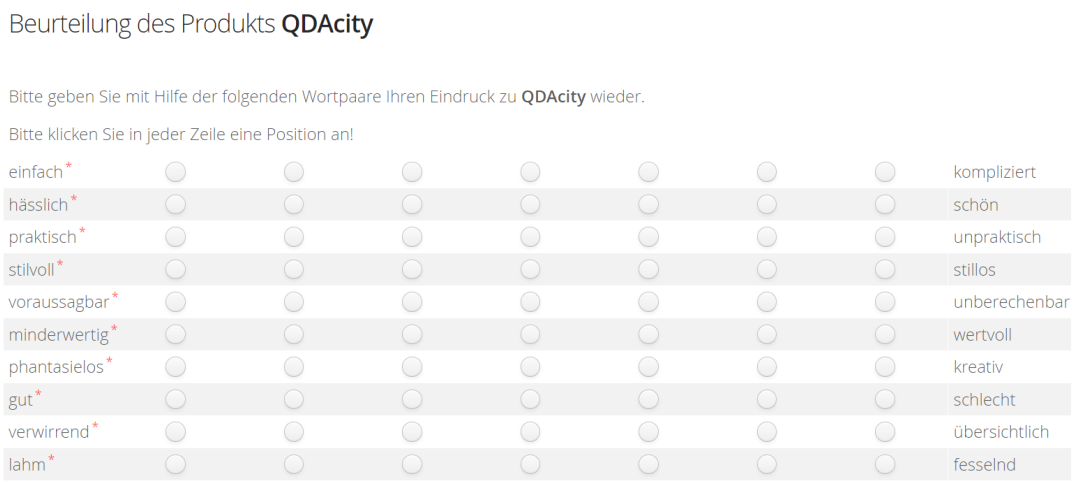

Abbildung 7.6: Kurzversion des AttrakDiff2 Fragebogens.

Zudem kann die Studie noch um Fragen erweitert werden, was genutzt wird, um

von den Teilnehmern weiteres gezielteres Feedback zu erhalten. Es wurden die folgenden Fragen hinzugefügt:

- Was hat Ihnen besonders gut am Tutorial System gefallen?
- Was hat Ihnen besonders schlecht am Tutorial System gefallen?
- Was würden Sie sich noch beim Tutorial System wünschen?

7.5 Ergebnisse

An der Studie haben insgesamt 10 Nutzer teilgenommen und im Folgenden werden die Ergebnisse vorgestellt. Diese lassen sich in die Auswertung des AttrakDiff-Short Fragebogens und der individuellen Fragen unterteilen.

7.5.1 AttrakDiff-Short Fragebogen

Die Daten, die durch die Beantwortung des AttrakDiff-Short Fragebogens erhoben wurden, können zunächst in die Ausprägung von hedonischer Qualität (HQ), pragmatischer Qualität (PQ) und Attraktivität (ATT) gegliedert werden. Dabei wurden jeweils vier Fragen zur Ermittlung der HQ und der PQ gestellt und zwei Fragen zur Ermittlung der ATT. In Abbildung 7.7 werden die Mittelwerte der einzelnen Ausprägungen gezeigt, wobei sich die PQ bei 0.53 Punkten, die HQ bei 0.65 Punkten und die ATT bei 1.50 Punkten befindet.

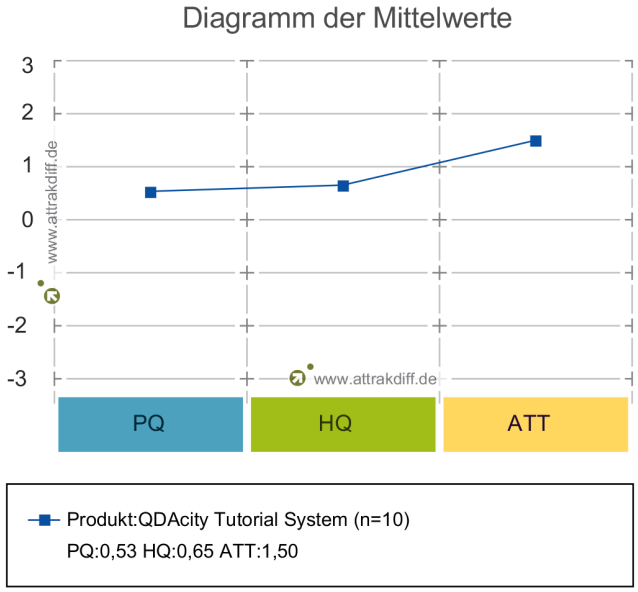

Abbildung 7.7: Ergebnisüberblick des Portfolios der UX Studie.

Dies zeigt, dass das Tutorial System zwar bereits über hedonische und pragmatische Qualitäten verfügt, es aber in dieser Hinsicht noch ausgebaut werden kann. Da die Mittelwerte aber alleine noch keine Einordnung des Tutorial Systems erlauben und auch nicht darstellen, wie unterschiedlich die Meinung der verschiedenen Teilnehmer ist, werden die Daten als Portfolio illustriert. Dabei wird die Ausprägung der HQ und der PQ ermittelt und einem oder mehreren Charakterbereichen zugewiesen. Das Konfidenz-Rechteck stellt dar, mit welcher Sicherheit das Tutorial System, mit den ermittelten HQ und PQ, einem Charakterbereich zugewiesen werden kann. Je größer das Konfidenz-Rechteck ist, desto unsicherer ist die Zuweisung und desto unterschiedlicher sind die Meinungen der Teilnehmer.

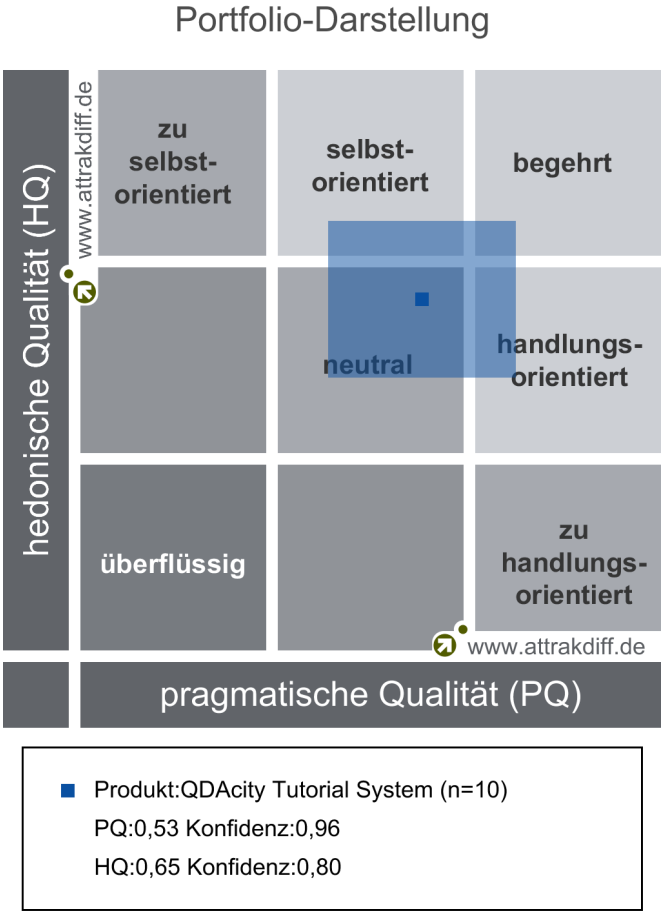

Abbildung 7.8: Ergebnisüberblick des Portfolios der UX Studie.

Wie die Abbildung 7.8 zeigt, kann das Tutorial System nicht eindeutig einem Charakterbereich zugeordnet werden, weil das Konfidenz-Rechteck über den Bereichen *neutral*, *selbst-orientiert*, *handlungs-orientiert* und *begehrt* liegt. Dabei ist eine Zuordnung zum Charakterbereich *begehrt* das optimale Ergebnis und das

langfristige Ziel. Im Fall des Tutorial Systems liegt die Zuordnung mit der höchsten Wahrscheinlichkeit beim *neutralen* Charakterbereich. Die Konfidenz liegt dabei bei 0.96 für die pragmatische Qualität und bei 0.80 für die hedonische Qualität. Dies zeigt, dass das Tutorial System nicht über eine extrem starke oder schwache Ausprägung der Qualitäten verfügt und diese Version des Tutorial Systems bereits in der Nähe des gewünschten Charakterbereiches liegt.

Neben dem Portfolio und den Mittelwerten können abschließend noch die einzelnen Wortpaare, die im Fragebogen verwendet wurden, analysiert werden. Dabei zeigt Abbildung 7.9, dass die Attraktivität des Tutorial Systems bereits stark ausgeprägt ist, jedoch die Wortpaare wie *lahm - fesselnd* in der HQ und wie *verwirrend - übersichtlich* in der PQ am schwächsten ausgeprägt sind und daher in der Weiterentwicklung besonders berücksichtigt werden können.

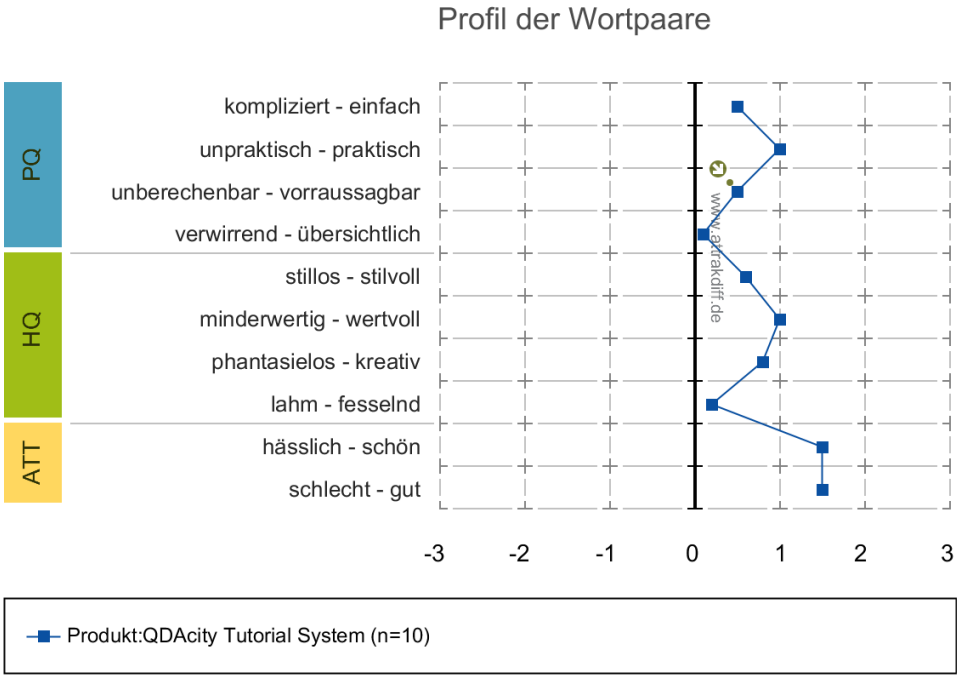

Abbildung 7.9: Profil der einzelnen Wortpaare im AttrakDiff-Short Fragebogen.

7.5.2 Individuelle Fragen

Durch die individuellen Fragen können die Teilnehmer Feedback in Textform zum Tutorial System geben. Dies ist besonders effektiv, um herauszufinden was Nutzern im Tutorial System gefällt, aber auch was diese stört oder vermissen. Durch dieses Feedback kann das Tutorial System dahingehend weiterentwickelt werden, dass eine bessere Nutzererfahrung ermöglicht wird. Zuerst wird aufgezeigt, was den Nutzern im Tutorial System gefallen hat:

- Klare Hinweise durch Hervorheben der Buttons
- Verschwimmen lassen der restlichen Benutzeroberfläche
- Beschreibung der einzelnen Schritte
- Kurze Länge des Tutorials
- Schritt für Schritt Erklärung
- Schlichtes Design ohne Überladen der Menüs

Die folgenden Aspekte haben den Nutzern nicht gefallen oder gefehlt und wären im Tutorial System wünschenswert gewesen:

- Fehlender Effekt bei einem neuen Schritt
- Verschwommene Benutzeroberfläche macht das Kennenlernen der UI unmöglich
- Hervorheben von Elementen ohne verschwommenen Hintergrund
- Keine Möglichkeit zu vorherigen Schritten zu gelangen
- Fehlende Nummerierung der Schritte
- Mehr Zwischenschritte, um die Verwendung von Codes zu erklären
- Popup-Tooltips für die einzelnen Menü-Elemente
- Bessere Auflösung mit größeren Elementen und größerer Schrift
- Weitere Tutorials für tiefere Benutzung

7.5.3 Zukünftige Arbeit

Nach der Durchführung der Studie konnten einige Optimierungen gefunden werden, die die Nutzer im Tutorial System begrüßen würden und daher in der weiteren Entwicklung umgesetzt werden sollen.

Zunächst sollen die nicht umgesetzten funktionalen Anforderungen FA 12 und FA 13 implementiert werden. Die FA 12 definiert, dass dem Nutzer angezeigt wird, wenn ein Schritt erfolgreich umgesetzt wurde. Dies soll mit Hilfe einer Animation in der Sidebar realisiert werden und so dem Nutzer eindeutig vermitteln, wann ein Schritt erfolgreich durchgeführt wurde. In FA 13 soll dem Nutzer das Ende eines Tutorials optisch dargestellt werden. Dies könnte durch ein Fenster verwirklicht werden, dass zur erfolgreichen Durchführung des Tutorial gratuliert und dem Nutzer noch weitere Tutorials vorschlägt, die dann sofort gestartet werden können. Denn bei der bisherigen Version haben Teilnehmer der Studie nicht gesehen, dass es noch weitere Tutorials gibt, und dies als negativ empfunden.

Neben den nicht umgesetzten funktionalen Anforderungen sollen weitere Schritte definiert werden, die vor allem die komplexen und schwer zu verstehenden Prozesse der Anwendung in noch kleineren Schritten für bessere und genauere Verständlichkeit abbilden soll. Der Prozess, wie Codes erstellt und angewendet werden, soll dabei priorisiert werden, da dieser Prozess einen sehr wichtigen Teil von QDAciy darstellt und die Teilnehmer der Studie gerade bei diesem Teil des Tutorials Kritik anmerkten. Auch gibt es viele Funktionen von QDAcity, die noch nicht als Schritt implementiert wurden und für Nutzer interessant sein könnten. Die bestehenden Tutorials werden dementsprechend angepasst, die neuen Schritte zu beinhalten und zu zeigen. Im Zuge der Anpassungen sollen auch die Beschreibungen der einzelnen Schritte überarbeitet werden, um den Nutzern den durchzuführenden Schritt optimal erläutern zu können.

Auch soll weitere Funktionalität in Form einer Übersicht von Statistiken zu den Tutorials implementiert werden. So soll sichtbar werden, wie viele Nutzer die einzelnen Basis Tutorials starten und wie viele von diesen das Tutorial auch vollständig durchführen. Dies kann genutzt werden, um Probleme bei den Tutorials erkennen zu können und gegebenenfalls Veränderungen vorzunehmen.

Optisch soll evaluiert werden, welche Schriftgröße bei dem Text des Tutorials für die Nutzer angenehmer wäre, um dies entsprechende umzusetzen.

8 Fazit

Ziel dieser Arbeit war die Implementierung eines Tutorial Systems in QDAcity, das mit Hilfe von Tutorials alle Bereiche der Anwendung Nutzern vorstellt und die Möglichkeit schafft, selbst Tutorials erstellen zu können.

Zunächst wurde die Anwendung QDActiy in Kapitel 2 vorgestellt und die verschiedenen Bereiche (Projektübersicht, Coding Editor und Kursübersicht) mit ihrer Funktionalität innerhalb der Anwendung beschrieben. Zudem wurde ein technischer Überblick über QDAcity, eine computerunterstützte qualitative Datenanalyse Software (CAQDAS), gegeben. In Kapitel 3 wurde dargestellt, welche Arten von Tutorials existieren, welche Vor- und Nachteile diese haben und welche Tutorialart deshalb am geeignetsten für das Tutorial System ist.

Insgesamt wurden 36 Anforderungen an das Tutorial System, 20 funktionale und 16 nicht-funktionale Anforderungen, in Kapitel 4 definiert. Um diese Anforderungen zu erfüllen, wurde eine Architektur des Tutorial Systems entwickelt, die in Kapitel 5 beschreibt, wie das Tutorial System im JavaScript Client und in der Web API umgesetzt werden soll.

Die Implementierung der Architektur in Kapitel 6 zeigt, wie das Tutorial System realisiert wurde. Es werden zuerst das erstellte Datenmodell und die REST Endpunkte vorgestellt und erklärt. Anschließend wird die Implementierung der Logik des Tutorial Systems in der Web API erläutert und wie die vordefinierten Daten initialisiert werden. Abschließend wird die Umsetzung des Tutorial Systems im JavaScript Client beschrieben und wie die Benutzeroberfläche gestaltet wurde.

Die Definition der Anforderungen aus Kapitel 4 wurde in Kapitel 7 genutzt, um die Implementierung des Tutorial Systems gegen diese zu evaluieren. Dabei konnten 33 der 36 definierten Anforderungen vollständig erfüllt werden, wobei für die drei nicht vollständig erfüllten Anforderungen die Implementierung der Benutzeroberfläche fehlt.

Neben der Evaluation der Anforderungen wurde eine UX Studie durchgeführt, bei welcher Teilnehmer zuerst ein Tutorial durchführen und anschließend einen Fragebogen ausfüllen. Dabei konnte ermittelt werden, dass das Tutorial System

bereits über einige pragmatische und hedonische Qualitäten verfügt und diese mit Hilfe des gesammelten Feedbacks in der Weiterentwicklung des Tutorial Systems noch verbessert werden können.

QDAcity verfügt nun über ein interaktives Tutorial System, das in die Anwendung integriert ist und von dem neue, aber auch erfahrene Nutzer profitieren können. Das Tutorial System ermöglicht es dem Nutzer vordefinierte, eigene oder fremde Tutorials durchzuführen, eigene Tutorials zu erstellen und diese zu teilen.

Literaturverzeichnis

Aberson, C. L., Berger, D. E., Healy, M. R., Kyle, D. J., & Romero, V. L. (2000). Evaluation of an Interactive Tutorial for Teaching the Central Limit Theorem. *Teaching of Psychology*, *27*(4), 289–291. https://doi.org/10.1207/S15328023TOP2704_08

Abrahao, S., Olsina, L., & Pastor, O. (2002). A methodology for evaluating quality and functional size of operative webapps. *Proc. of 2nd International Workshop on Web Oriented Software Technology*, 1–20.

Gong, J., Lu, J., & Cai, L. (2016). An Induction to the Development of Software Quality Model Standards. *2016 Third International Conference on Trustworthy Systems and their Applications (TSA)*, 117–122. https://doi.org/10.1109/TSA.2016.28

Hassenzahl, M., Burmester, M., & Koller, F. (2003). AttrakDiff: Ein Fragebogen zur Messung wahrgenommener hedonischer und pragmatischer Qualität. In G. Szwillus & J. Ziegler (Hrsg.), *Mensch Computer 2003: Interaktion in Bewegung* (S. 187–196). B. G. Teubner.

Hassenzahl, M., Koller, F., & Burmester, M. (2008). Der User Experience (UX) auf der Spur: Zum Einsatz von www.attrakdiff.de. In H. Brau, S. Diefenbach, M. Hassenzahl, F. Koller, M. Peissner & K. Röse (Hrsg.), *Tagungsband UP08* (S. 78–82). Fraunhofer Verlag.

ISO/IEC 25010. (2011). *ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.*

Käfer, V., Kulesz, D., & Wagner, S. (2017). What Is the Best Way For Developers to Learn New Software Tools? An Empirical Comparison Between a Text and a Video Tutorial. *The Art, Science, and Engineering of Programming*, *1*(2). https://doi.org/10.22152/programming-journal.org/2017/1/17

Kaufmann, A. (2021). *Domain Modeling Using Qualitative Data Analysis* (Diss.). Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU).

Kawaf, F. (2019). Capturing digital experience: The method of screencast videography. *International Journal of Research in Marketing*, *36*(2), 169–184. https://doi.org/https://doi.org/10.1016/j.ijresmar.2018.11.002

Laugwitz, B., Held, T., & Schrepp, M. (2008). Construction and Evaluation of a User Experience Questionnaire. *USAB 2008*, *5298*, 63–76. https://doi.org/10.1007/978-3-540-89350-9_6

Lloyd, S. A., & Robertson, C. L. (2012). Screencast Tutorials Enhance Student Learning of Statistics. *Teaching of Psychology*, *39*(1), 67–71. https://doi.org/10.1177/0098628311430640

Mayer, R. E. (2009). The Promise of Multimedia Learning. In *Multimedia Learning* (2. Aufl., S. 3–27). Cambridge University Press. https://doi.org/10.1017/CBO9780511811678.003

McNeely, I., & Wolverton, L. (2008). *Reinventing Knowledge: From Alexandria to the Internet*. W. W. Norton. https://books.google.de/books?id=jBzY0nQe7UIC

Mısırlı, A., Bener, A., Çağlayan, B., Çalıklı, G., & Turhan, B. (2014). Field studies: a methodology for construction and evaluation of recommendation systems in software engineering. In M. P. Robillard, W. Maalej, R. J. Walker & T. Zimmermann (Hrsg.), *Recommendation Systems in Software Engineering* (S. 329–355). Springer. https://doi.org/10.1007/978-3-642-45135-5_13

Musch, O. (2021a). Observer. In *Design Patterns mit Java: Eine Einführung* (S. 41–59). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-35492-3_5

Musch, O. (2021b). Singleton. In *Design Patterns mit Java: Eine Einführung* (S. 23–31). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-35492-3_3

Pederson, J. R. (2017). The SAGE Encyclopedia of Communication Research Methods [pages 835-838]. https://doi.org/10.4135/9781483381411

Roto, V., Obrist, M., & Väänänen-Vainio-Mattila, K. (2009). User experience evaluation methods in academic and industrial contexts. *Proceedings of the Workshop UXEM*, *9*, 1–5.

Singh, C. (2008). Interactive learning tutorials on quantum mechanics. *American Journal of Physics*, *76*(4), 400–405. https://doi.org/10.1119/1.2837812

Thelen, T., Lucke, R., & Siekmeyer, A. (2013). Tutonium – Interaktive Tutorials für Web-Anwendungen. In A. Breiter & C. Rensing (Hrsg.), *DeLFI 2013: Die 11 e-Learning Fachtagung Informatik* (S. 215–226). Gesellschaft für Informatik e.V.

Thomaschewski, J., Hinderks, A., & Schrepp, M. (2018). Welcher UX-Fragebogen passt zu meinem Produkt? In S. Hess & H. Fischer (Hrsg.), *Mensch und Computer 2018 - Usability Professionals* (S. 437–446). Gesellschaft für Informatik e.V. Und German UPA e.V. https://doi.org/10.18420/muc2018-up-0150