Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät, Department Informatik

Lixian Lao

BACHELORARBEIT

Userfeedback-System für QDAcity

Submitted on 8.11.2023.

Supervisor:
Prof. Dr. Dirk Riehle, M.B.A.
Julia Mucha M.Sc.
Professur für Open-Source-Software
Department Informatik, Technische Fakultät
Friedrich-Alexander University Erlangen-Nürnberg

Versicherung

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.

_____

Nürnberg, 7.11.2023

License

_____

Nürnberg, 7.11.2023

Abstract

In this digital world where web application has become a crucial part of our daily lives, the importance of user feedback has become apparent. User feedback is important in many aspects, since it can help identifying pain points and give one a new perspective of their product and that's why every website should give the user ways to provide various kinds of feedbacks. With the introduction of user feedback, so-called feedback loops are created which will help requirements engineers to successfully tailor the application to the users need. Additionally, the idea of continuous improvement can used to meet the users' expectations and improve their loyalty towards the application. In this thesis, we will analyze different types of user feedback and then design and implement a feedback system for the cloud-based web application QDAcity.

Zusammenfassung

In dieser digitalen Welt, in der Webanwendungen zu einem wichtigen Teil unseres täglichen Lebens geworden sind, ist die Bedeutung von Nutzerfeedback offensichtlich geworden. Nutzerfeedback ist in vielerlei Hinsicht wichtig, da es dabei helfen kann, Schmerzpunkte zu identifizieren und eine neue Perspektive auf das Produkt zu gewinnen. Deshalb sollte jede Website den Nutzern die Möglichkeit geben, verschiedene Arten von Feedback zu geben. Mit der Einführung von Benutzer-Feedback werden sogenannte Feedback-Schleifen geschaffen, die den Anforderungsingenieuren helfen, die Anwendung erfolgreich auf die Bedürfnisse der Benutzer zuzuschneiden. Darüber hinaus kann die Idee der kontinuierlichen Verbesserung genutzt werden, um die Erwartungen der Benutzer zu erfüllen und ihre Loyalität gegenüber der Anwendung zu erhöhen. In dieser Arbeit werden verschiedene Arten von Nutzerfeedback analysiert und anschließend ein Userfeedback-System für die cloudbasierte Webanwendung QDAcity entworfen und implementiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Codeverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	**Application Programming Interface**
UML	**Unified Modeling Language**
QDA	**Qualitative Datenanalyse**
JS	**JavaScript**
FA	**Funktionale Anforderung**
CRUD	**Create Read Update Delete**
CI	**Continuous Improvement**
NPS	**Net Promoter Score**
CSAT	**Customer Satisfaction Score**
WTP	**Willingness To Pay**
REST	**Representational State Transfer**
GAE	**Google App Engine**
HTTP	**Hypertext Transfer Protocol**

1 Einleitung

Kaizen (Kai: „Veränderung“, zen: „Zum Besseren“) ist ein Begriff aus dem japanischen und beschreibt die kontinuierliche und permanente Verbesserung von Tätigkeiten, Produkten und Prozessen. In der heutigen Wirtschaft ist Kaizen, auch bekannt als den kontinuierlichen Verbesserungsprozess, ein grundsätzlicher Bestandteil des Qualitätsmanagements eines jeden erfolgreichen Unternehmens geworden. Dieser Ansatz, der ursprünglich in der Autoindustrie eingesetzt wurde, hat es Unternehmen wie Toyota ermöglicht, ihren Produktionsprozess schneller, günstiger und effizienter zu gestalten[1]. Heutzutage gibt es viele Werkzeuge und Techniken, die genutzt werden können, um den Ansatz von kontinuierlicher Verbesserung zu unterstützen. Insbesondere Userfeedback ist einer der wichtigsten Werkzeuge, vor allem hinsichtlich Webanwendungen. Denn in diesen digitalen Zeiten, wo Menschen im Durschnitt fast 3 Stunden täglich damit verbringen, online zu sein[2], spielen Webanwendungen eine große Rolle in unserem Alltag. Da alle Anwendungen um die Loyalität, Zufriedenheit und Aufmerksamkeit der Benutzer konkurrieren ist es wichtig, dass sich eine Webanwendung kontinuierlich verbessert, um zufriedenstellend zu bleiben (Lehmann, 1974). Im Prozess der Verbesserung spielen die Meinungen und Bedürfnisse der Benutzer und Kunden eine entscheidende Rolle. Durch die Integration von Userfeedbacks sollen Feedbackloops entstehen, die langfristig dabei helfen, die Anwendung kontinuierlich zu verbessern.

1.1 Ziel der Arbeit

In dieser Arbeit soll ein Userfeedback-System für die Webanwendung QDAcity konzeptioniert und implementiert werden. Das System soll die Daten der Feedbacks zusammenfassen und graphisch auf der Admin-Page anzeigen. Somit bekommen die Administratoren einen Überblick, wie zufrieden die Nutzer mit dem aktuellen Stand der Anwendung sind. Dadurch sollen mögliche Verbesserungen und Pain-Points frühzeitig erkannt und gelöst werden.

[1] https://global.toyota/en/company/vision-and-philosophy/production-system/

[2] https://de.statista.com/statistik/daten/studie/1073570/umfrage/taegliche-nutzungsdauer-des-medialen-internets-in-deutschland/

2 QDAcity

Im folgenden Kapitel werden die wichtigsten Seiten und Funktionen von der Webanwendung QDAcity[3] erklärt.

2.1 Funktionalität

QDAcity ist eine cloudbasierte Webanwendung für die qualitative Datenanalyse. Sie hilft Benutzern dabei relevante Informationen aus einer Menge von Daten zu strukturieren und zu speichern. Im Folgenden werden die verschiedenen Kernfunktionen und Technologien von QDAcity beschrieben.

2.1.1 Personal-Dashboard

Das Personal-Dashboard dient als zentrale Übersicht von Projekten, Kursen und neuen Funktionen. Hier kann der Benutzer Projekte und Kurse[4] erstellen, löschen und bearbeiten. Darüber hinaus wird er im mittleren Bereich über die neusten Änderungen der Anwendung informiert. Abbildung 2.1 zeigt den Aufbau des Personal-Dashboards.

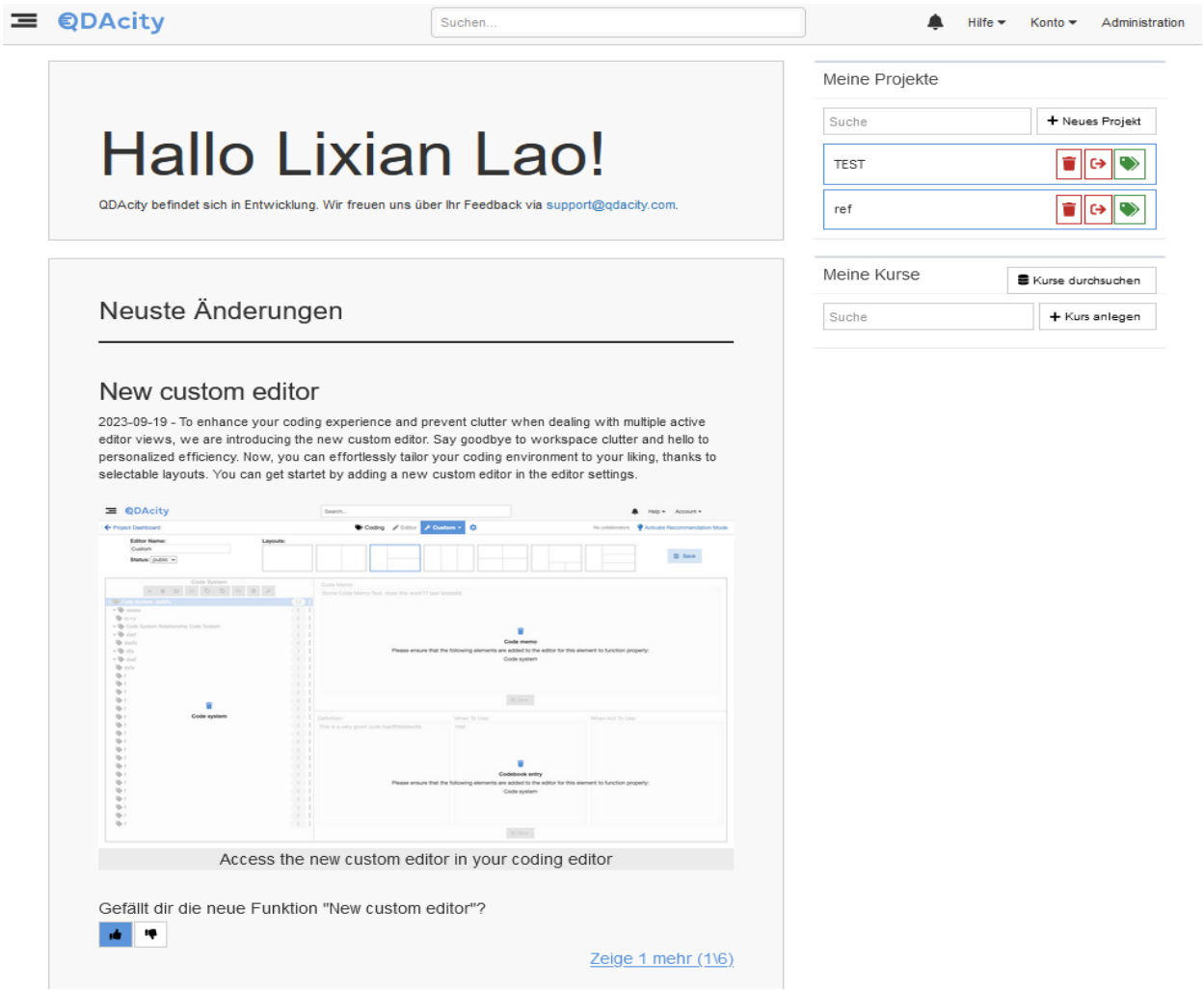

Abbildung 2.1 Personal-Dashboard von QDAcity

[3] https://qdacity.com/de/
[4] https://qdacity.com/de/qda-lehre/

2.1.2 Public-Pages

QDAcity dient nicht nur als Hilfstool der qualitativen Datenanalyse, sondern auch als Informationsseite verschiedener QDA-Themen. Diese sogenannten Public-Pages, wie in Abbildung 2.2 abgebildet, sind für alle Nutzer, egal ob registriert/eingeloggt oder nicht, zugänglich und geben einen Einblick in die Prozesse, Methoden und Best-Practices der qualitativen Datenanalyse. Zusätzlich geben sie dem Nutzer einen Überblick über die verschiedenen Funktionen von QDAcity, wie zum Beispiel dem Transkription-Dienst, welcher automatisch Audiodateien in Text umwandelt.

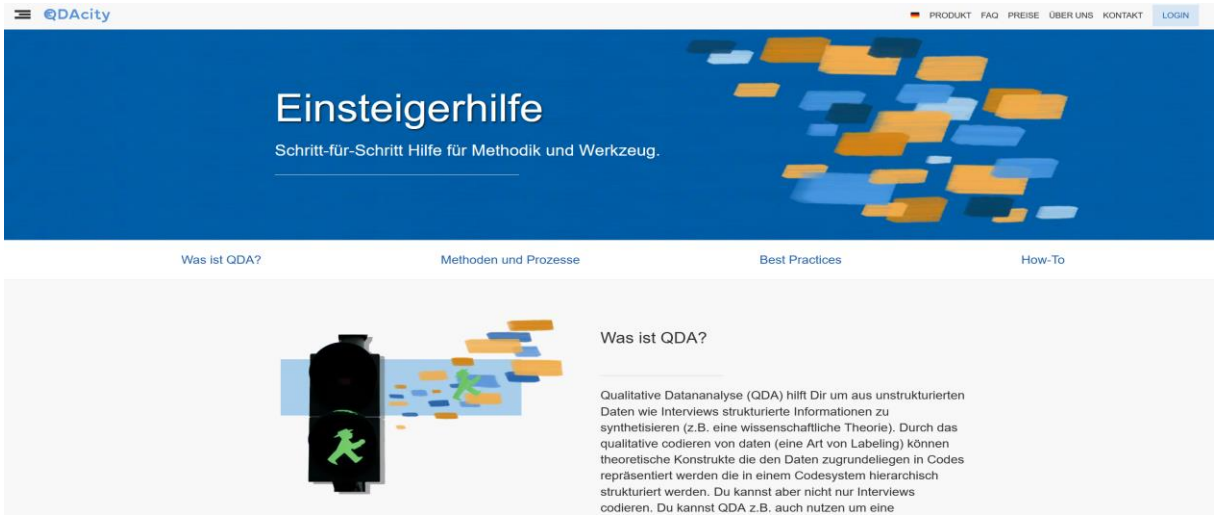

Abbildung 2.2 Public-Page von QDAcity

2.1.3 Coding-Editor

Der Coding-Editor ist die Kernfunktion von QDAcity. Dort können Dokumente, Bilder oder Ähnliches hochladen werden und diese dann mit Hilfe der gegebenen Tools auf der linken Seite codiert[5], strukturiert und erweitert werden (siehe Abbildung 2.3). Das ist die Seite, auf der sich die User mit hoher Wahrscheinlichkeit am längsten aufhalten.

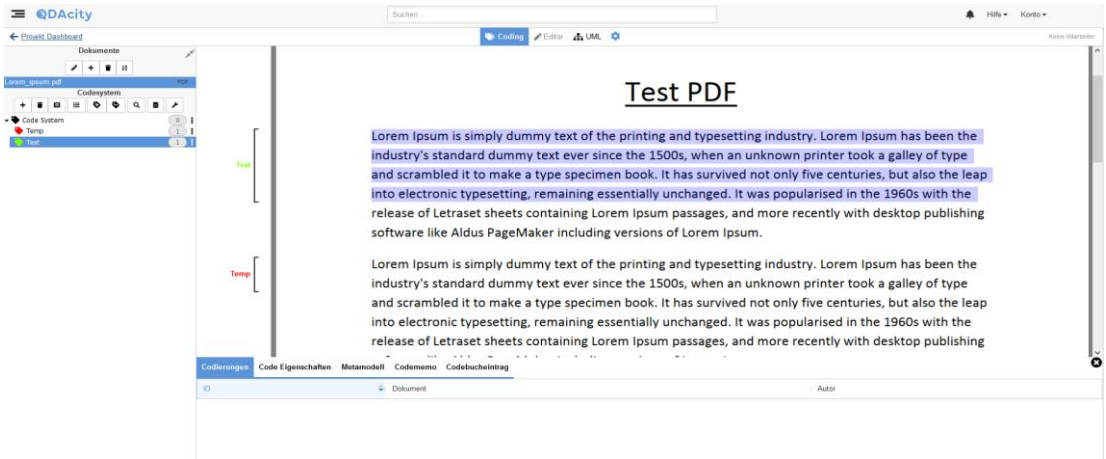

Abbildung 2.3 Coding-Editor von QDAcity

[5] https://qdacity.com/de/qualitatives-codieren/

2.1.4 Admin-Page

Die Admin-Page ermöglicht es Administratoren, die aktuellen Statistiken der Anwendung einzusehen. Diese werden aktuell in 4 Kategorien unterteilt (siehe Abbildung 2.4). Zu jeder dieser Kategorien gibt es allgemeine Statistiken, die sich ganz oben auf der Seite befinden.

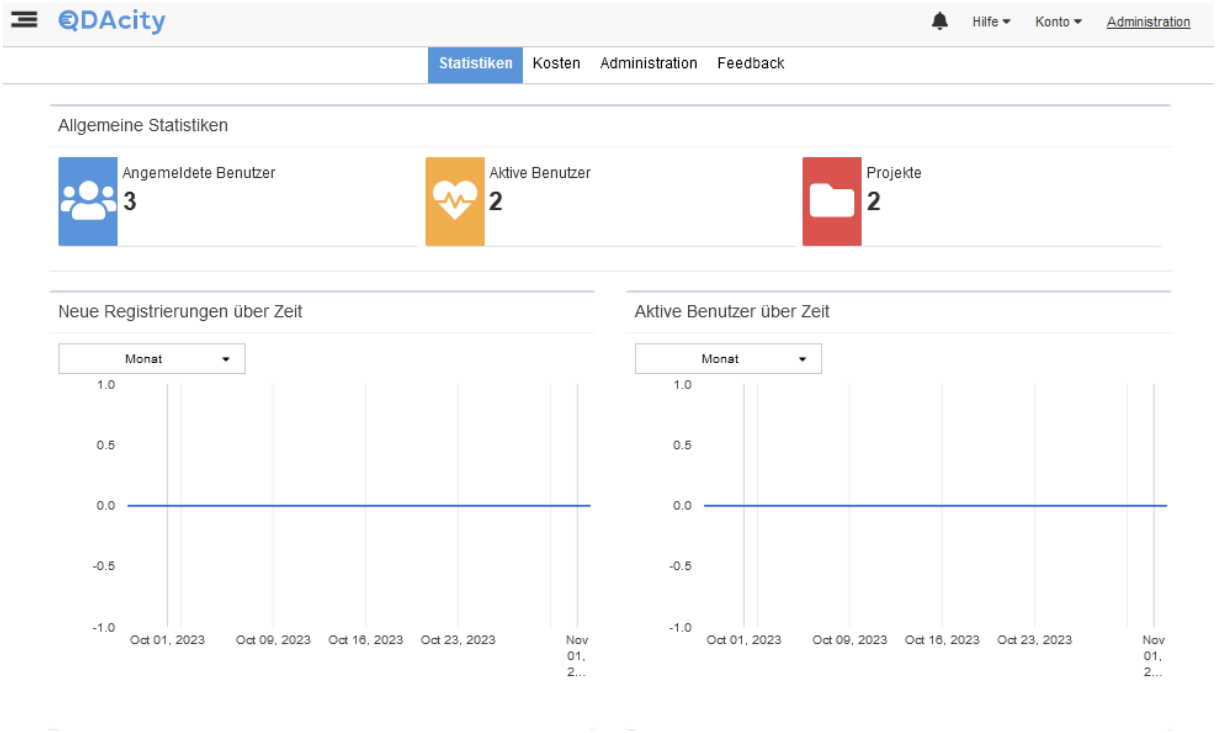

Abbildung 2.4 Admin-Page von QDAcity

Statistiken

In dieser Kategorie werden allgemeine Statistiken wie die Anzahl angemeldeter und aktiver Nutzer sowie die Anzahl an Projekten, angezeigt. Für eine graphische Darstellung werden Diagramme benutzt, bei denen man den Zeitraum der Diagramme mit einem Dropdownmenü ändern kann.

Kosten

Hier lassen sich die monatlichen Kosten für die verschiedenen Dienste, wie App Engine und Cloud Storage einsehen.

Administration

Zu dem Zeitpunkt dieser Arbeit hat diese Kategorie noch keine Funktionalitäten implementiert.

Feedback

Diese Kategorie wurde im Zuge dieser Arbeit hinzugefügt und soll die aggregierten Daten der Feedback-Komponenten beinhalten. Hier sollen die Daten durch Diagramme dargestellt werden.

2.2 Technologie

QDAcity ist eine cloudbasierte Webanwendung, welche auf dem Client-Server-Konzept beruht. Sie besteht aus einem JavaScript Frontend und einem auf Java 8[6] basiertem Backend. Außerdem benutzt die Anwendung das Prinzip der RESTful Webservices[7], um zwischen Frontend und Backend via HTTP-Requests zu kommunizieren.

2.2.1 Frontend

Im Frontend wird die JavaScript-Bibliothek React[8] zur Erstellung der Benutzeroberfläche verwendet. React ermöglicht es, die Benutzeroberfläche in kleine individuelle wiederverwendbare Komponenten zu unterteilen. In Kombination mit React wird die syntaktische Erweiterung JavaScript XML, auch JavaScript Syntax Extension[9] (JSX) genannt, benutzt, um den Code simpler und damit performanter zu halten. JSX bietet die Möglichkeit, JS-Code in einer HTML ähnlicher Form zu schreiben, was für die meisten Entwickler einfacher zu verstehen und nutzen ist.

2.2.2 Backend

Das Backend basiert auf Java 8 und stellt eine RESTful API zur Verfügung, die für das Erstellen, Lesen, Aktualisieren und Löschen (CRUD) entsprechende REST-Endpunkte anbietet. Die Java Anwendung wird über die Platform-as-a-Service (PaaS)-Lösung Google App Engine[10] (GAE) gehostet, welche es ermöglicht, die Anwendung in der Cloud zu betreiben.

Als Speicherung wird die NoSQL-Dokumentdatenbank Firestore im Datastore Modus[11] verwendet. Der Zugriff auf die Daten erfolgt mit der Open-Source Java data access API Objectify[12], die spezifisch für die GAE erstellt wurde.

[6] https://www.java.com/de/download/help/java8_de.html
[7] https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html
[8] https://react.dev/
[9] https://react.dev/learn/writing-markup-with-jsx#
[10] https://cloud.google.com/appengine#
[11] https://cloud.google.com/datastore/docs?hl=de
[12] https://github.com/objectify/objectify/wiki

3 Userfeedback

Userfeedback setzt sich aus den Begriffen User (engl. für Benutzer) und Feedback (engl. für Rückmeldung) zusammen und beinhalten Daten zu Präferenzen und Eindrücken, die ein Benutzer zu einem Produkt oder zu einer Anwendung hat. Sie werden gesammelt, um dem Produktteam einen besseren Überblick zu geben, was die Kunden von ihrem Produkt halten und was für Verbesserungsvorschläge die Kunden in der Anwendung oder dem Produkt sehen. Durch das Sammeln solcher Feedbacks, wird der Benutzer in den Entwicklungsprozess einer Anwendung involviert und fühlt sich so der Anwendung viel näher, was Loyalität aufbaut. Zusätzlich werden durch das Einführen von Benutzerfeedback sogenannte Feedback Loops erzeugt, die die kontinuierliche Verbesserung (CI) eines Projekts vorantreibt.

3.1 Kontinuierliche Verbesserung & Feedback Loops

Kontinuierliche Verbesserung ist der Prozess, der ständigen Änderung an einem Produkt oder einer Dienstleistung basierend auf Ergebnissen der Analyse von Benutzerfeedback. Dieser Prozess kann dabei helfen, die Qualität, Leistung und Benutzerzufriedenheit einer Anwendung zu verbessern. Die Verbesserung geschieht dabei in kleinen und iterativen Zyklen, in denen die Lösungen getestet und weiter verbessert werden. Wie so ein Zyklus aussehen kann, wird in der Abbildung 3.1 dargestellt.

In dem PDCA-Zyklus, auch Deming-Zyklus[13] genannt, gibt es vier Schritte. Angefangen mit dem „Plan“ Schritt, wird ein Problem oder Feature analysiert und es wird ein Plan erstellt, um das Problem zu lösen oder das Feature zu implementieren. Im „Do“ Schritt wird der Plan umgesetzt und Feedback dazu gesammelt. Im darauffolgenden „Check“ Schritt wird das Feedback analysiert, bewertet und darauf geprüft, ob es den Erwartungen aus der Planphase entspricht. Im letzten „Act“ Schritt wird die Implementierung dann mit den Ergebnissen aus dem „Check“ Schritt angepasst und verbessert. Anschließend wird der Zyklus entweder mit dem gleichen Problem, das noch verbessert werden muss, erneut gestartet, oder es wird ein neues Problem oder Feature angegangen. Mit jedem Zyklusdurchlauf wird damit ein neuer und besserer Standard der Anwendung implementiert (Schuster, 2022).

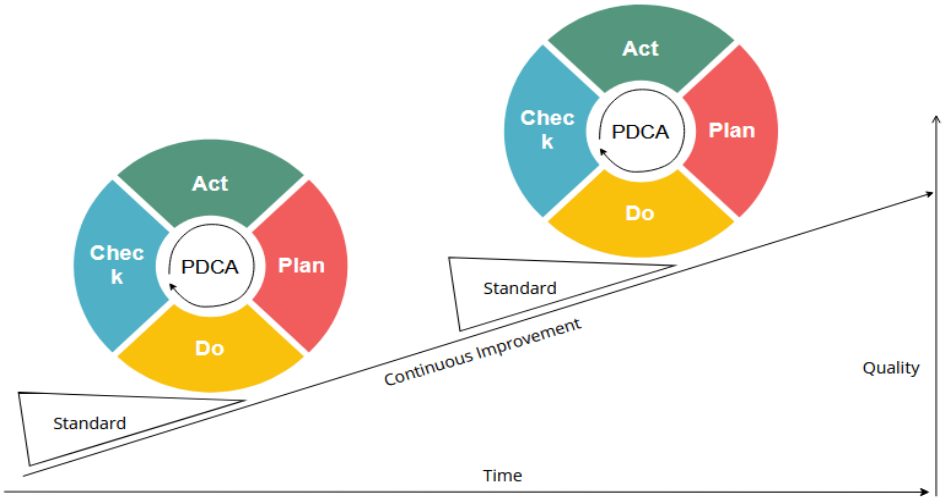

Abbildung 3.1 Zusammenhang zwischen Kontinuierlicher Verbesserung und Feedback Loops

[13]https://t2informatik.de/wissen-kompakt/pdca-zyklus/

3.2 Arten von Feedback

Auf der obersten Ebene kann Feedback in Quantitatives und Qualitatives Feedback kategorisiert werden. Im Folgenden werden beide Kategorien erklärt und darauf eingegangen, wie diese gesammelt werden können.

3.2.1 Quantitatives Feedback

Bei quantitativem Feedback handelt es sich um Daten, die in quantitativer oder numerischer Form, wie Zahlen oder anderen messbaren Größen, dargestellt werden. Diese Art von Feedback ist in der Regel eindeutig und einfach zu analysieren und wird mithilfe von Metriken gewonnen (Haije, 2017).

3.2.1.1 Net Promoter Score

Der sogenannte Net Promoter Score (NPS) beschreibt das Verhältnis zwischen Kunden, die das Gesamtprodukt weiterempfehlen würden und Kunden, die das Produkt kritisieren würden. Die Abfrage dieses Feedbacks geschieht in Form einer Likert-Skala (Likert, 1932) mit fünf bis zehn Stufen und einer einzelnen Frage: „Wie wahrscheinlich ist es auf einer Skala von eins bis zehn, dass Sie unser Produkt einer Freundin/einem Freund weiterempfehlen?“ (siehe Abbildung 3.2).

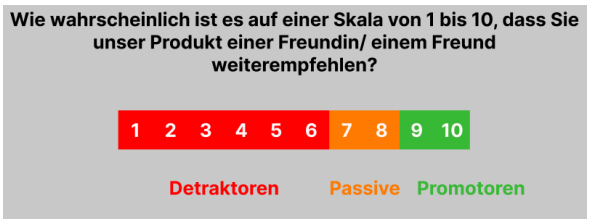

Abbildung 3.2 Beispiel für eine NPS-Abfrage

Bei der Berechnung des NPS werden Kunden, die mit einer sechs oder weniger geantwortet haben als Detraktoren (Rot), die mit sieben oder acht geantwortet haben als Passive oder Indifferente (Orange) und die, die mit einer neun oder zehn geantwortet haben als Promotoren (Grün) eingestuft. Die Berechnung erfolgt dann anhand dieser Formel:

$$NPS = Promotoren(\%) - Detraktoren(\%)$$

Beispiel: 200 Befragte, 130 Promotoren, 40 Passive, 30 Detraktoren

$$NPS = \frac{130}{200} * 100 - \frac{30}{200} * 100 = 50$$

Der Wertebereich des NPS liegt also zwischen -100 und 100. Mit diesem Score können nun Schlussfolgerungen gezogen werden, ob die Kunden zufrieden oder unzufrieden mit dem Produkt oder der Marke sind. Sie gilt als einer der besten Indikatoren, um die Loyalität der Kunden zu schätzen, denn durch das Weiterempfehlen geben Sie nicht nur eine gute Leistung des Produkts an, sondern setzen auch ihren eigenen Ruf aufs Spiel (Reichheld, 2003).

3.2.1.2 Customer Satisfaction Score

Der Customer Satisfaction Score (CSAT) beschäftigt sich mit der Zufriedenheit des Kunden vor, während oder nach einer Aktion. Diese Art von Feedback ist vergleichbar zu dem NPS aufgebaut und besteht ebenfalls aus nur einer Frage und einer fünf bis zehnstufigen Likert-Skala (siehe Abbildung 3.3).

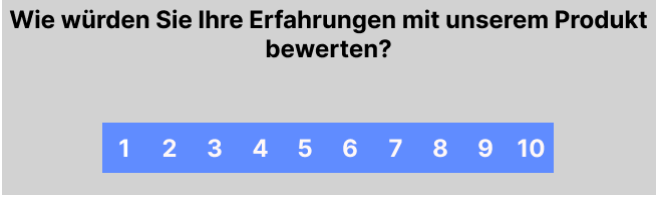

Abbildung 3.3 Beispiel für CSAT-Abfrage

Für die Berechnung des CSAT-Scores werden alle Antworten addiert und durch die Anzahl der Antworten geteilt. Mit Hilfe des Scores kann nun die aktuelle Kundenzufriedenheit über das Gesamt- oder Teilprodukt bestimmt werden. Die hohe Kundenzufriedenheit spielt eine zentrale Rolle in der Entwicklung eines Unternehmens, weil diese zu einer erhöhten Nachfrage führt, die wiederum zum erhöhten Profit und Unternehmenswachstum führt (Kotler & Keller, 2006).

3.2.1.3 Engagement-Metrik

Die Engagement-Metrik wird benutzt, um Interaktionen der Benutzer mit einer Website zu tracken (Hoffman & Fodor, 2010). Diese Metrik ist hilfreich, um den Erfolg von Inhalten und Produkten zu bewerten. Bekannte Vertreter dieser Metrik sind die von Social-Media-Plattformen bekannten Like/Dislike Buttons und Seitenaufrufe auf einer Webseite.

3.2.1.4 Willingness-To-Pay

Die Willingness-To-Pay-Metrik (WTP) wird benutzt, um den maximalen Wert zu bestimmen, den ein Benutzer für ein Produkt oder einer Dienstleistung bereit ist zu zahlen (Diller, 2008). Sie ist besonders nützlich, um den optimalen Preis herauszufinden, der die Anzahl an zufriedenen Kunden maximiert und gleichzeitig den maximalen Profit erzeugt. Sie wird meistens in Form von Umfragen gesammelt.

3.2.2 Qualitatives Feedback

Bei qualitativem Feedback handelt es sich um konkrete Meinungen oder Ideen eines Benutzers hinsichtlich eines Produktes oder einer Dienstleistung (Haije, 2017). Diese Kategorie von Feedback wird meistens im Zusammenhang mit einem quantitativen Feedback in Form eines Textfeldes gesammelt und wird in Rahmen dieser Arbeit auch so genutzt.

3.3 Auswahl

Aufgrund der starken Weiterentwicklung und der daraus folgenden möglichen Änderungen der UI und Funktionalitäten, wurde bei der Auswahl der Komponenten vor allem die Hauptfunktionen, wie Coding-Editor und Informationsseiten (Public-Pages) in Betracht gezogen. Anstatt eine Metrik wie NPS hinzuzufügen, welche die allgemeine Erfahrung der Anwendung sammelt, wurden Metriken wie Engagement und CSAT in Betracht gezogen, weil diese an spezifischeren Orten und Funktionen, Feedback sammeln. So sollen Like und Dislike Buttons als Engagement-Metrik an das Ende der Informationsseite, so wie bei jeder neuen Änderung im Personal-Dashboard eingefügt werden. Für den Coding-Editor soll CSAT eingefügt werden, um herauszufinden, wie zufrieden der Benutzer mit der Kernfunktion der Anwendung ist. Um mehr Informationen über die Zufriedenheit der Benutzer im Zusammenhang mit dem Coding-Editor zu erhalten, soll es dem Benutzer zusätzlich zu CSAT noch möglich sein, ein qualitatives Feedback in Form eines Textes abzugeben.

4 Anforderungen

4.1 Funktionale Anforderungen

Im Folgenden werden die Funktionale Anforderungen (FA) für die verschiedenen Feedback-Komponenten definiert. Sie werden nach der FunktionsMASTER Satzschablone (Die SOPHISTen, 2016), siehe Abbildung 4.1, definiert.

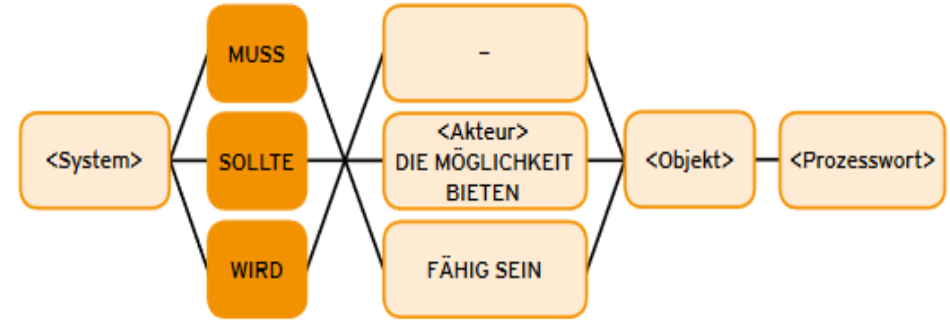

Abbildung 4.1 FunkitonsMASTER Satzschablone

Das <System> bezieht sich in diesem Fall immer auf eine einzelne Komponente und mit den Schlüsselwörtern MUSS, SOLLTE, WIRD wird die rechtliche Verbindlichkeit festgelegt. Verpflichtende Anforderungen werden mit dem Schlüsselwort MUSS gekennzeichnet. Wunschanforderungen, die nicht unbedingt erfüllt werden müssen, werden mit SOLLTE gekennzeichnet. Mit WIRD werden zukünftige Anforderungen formuliert, die nicht zum aktuellen Zeitpunkt umgesetzt werden müssen. Dennoch müssen sie bei der Umsetzung berücksichtigt werden, auch wenn sie noch nicht getestet werden können.

4.1.1 Like/Dislike-Komponente

FA 1: Die Like/Dislike-Komponente muss die Zugehörigkeit des Feedbacks anzeigen.

FA 2: Die Like/Dislike-Komponente muss die Möglichkeit bieten einen Button anzuklicken.

FA 3: Die Like/Dislike-Komponente sollte 3 Zustände haben (Liked/Disliked/NotSelected).

FA 4: Die Like/Dislike-Komponente muss fähig sein, den aktuellen Zustand farbig zu repräsentieren.

FA 5: Die Like/Dislike-Komponente sollte fähig sein, den aktuellen Zustand nach einem neu laden der Seite korrekt anzuzeigen.

4.1.2 Customer Satisfaction Score & Text-Komponente

FA 6: Die CSAT & Text-Komponente muss die Möglichkeit bieten Feedback mit einem Button klick zu senden.

FA 7: Die CSAT & Text-Komponente muss die Zugehörigkeit des Feedbacks anzeigen.

FA 8: Die CSAT & Text-Komponente muss eine Rückmeldung geben, wenn ein Feedback versendet worden ist.

FA 9: Die CSAT & Text-Komponente soll dem Benutzer eine Möglichkeit geben, die Komponente auszublenden.

4.1.3 Admin-Page-Komponenten

FA 10: Die Admin-Page-Komponenten müssen die richtigen Feedbacks aus dem Backend holen.

FA 11: Die Admin-Page-Komponenten müssen dem Administrator ermöglichen, zwischen verschiedenen Feedbackarten zu wechseln.

FA 12: Die Admin-Page-Komponenten müssen dem Administrator eine graphische Darstellung passend zum Feedbacktyp zeigen.

FA 13: Die Admin-Page-Komponenten müssen dem Administrator allgemeine Statistiken, wie Gesamte Like/Dislike oder CSAT-Score anzeigen.

4.2 Nicht-Funktionale Anforderungen

Die Nicht-Funktionalen Anforderungen (NFA) werden nach den Qualitätskriterien der ISO 25010[14] definiert. ISO 25010 ist die internationale Norm für Qualitätskriterien von Software, IT-Systemen und Software-Engineering (*System and Software Quality Requirements and Evaluation*). Im Folgenden werden die Nicht-Funktionalen Anforderungen für das Userfeedback-System, wie in der Abbildung 4.2 gezeigt nach acht Kernkriterien für die Softwarequalität definiert.

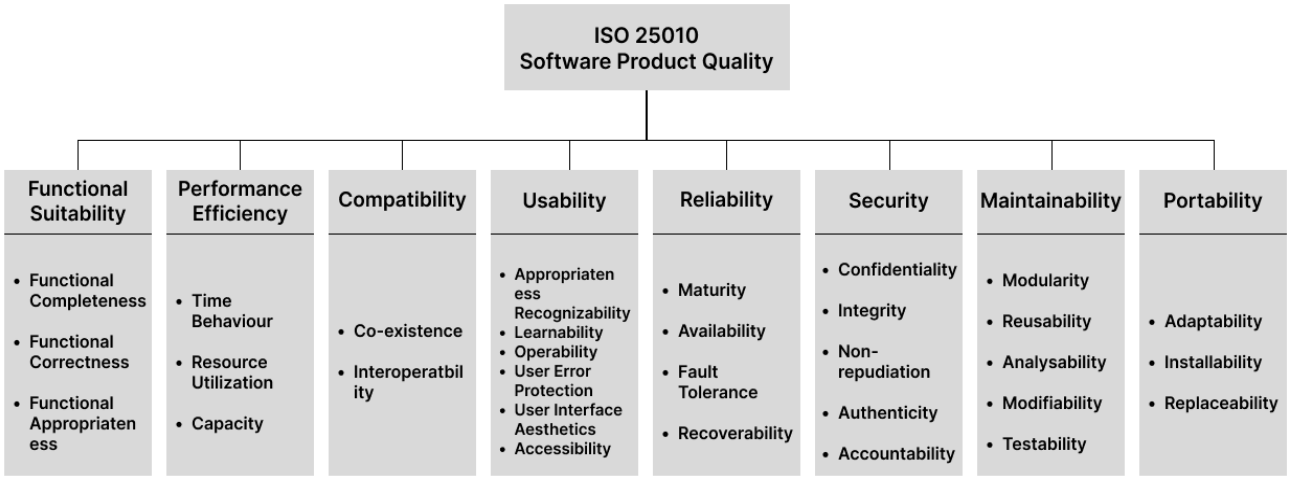

Abbildung 4.2 Qualitätskriterien nach ISO 25010

4.2.1 Funktionalität

NFA 1: Das System soll keine existierenden Funktionen von QDAcity einschränken oder beeinträchtigen.

4.2.2 Kompatibilität

NFA 2: Das System muss mit den verwendeten Diensten (Google App Engine, Datastore, Cloud Endpoints) kompatibel sein.

4.2.3 Benutzbarkeit

NFA 3: Das System ist an das Farbschema von QDAcity angepasst.

NFA 4: Das System soll Tastaturbedienung unterstützen.

[14] https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

4.2.4 Sicherheit

NFA 5: Das System muss bei jedem Endpunkt-Aufruf überprüfen, ob der Anwender autorisiert ist, diese zu benutzen.

4.2.5 Wartbarkeit

NFA 6: Das System ist mit entsprechenden Tests abgedeckt (Unittests, E2E Test, Integrationstests).

NFA 7: Die Komponenten sind modular und können wiederverwendetet werden.

5 Architektur

Im folgenden Kapitel wird die Architektur des Userfeedback-Systems beschrieben. Dafür wird zunächst die Architektur auf einer abstrakten Ebene betrachtet. Im Zuge dessen werden die verschiedenen Aktionen, die die Nutzer und Administratoren durchführen können, in Form eines Use-Case-Diagramms erläutert. Danach folgt die Beschreibung der Feedbackarten, die eingefügt werden können anhand eines UML-Datenmodells. Anschließend folgt eine Übersicht über die aktuelle Architektur im Backend von QDAcity und abschließend wird der Ablauf der Frontend-Komponenten erklärt.

5.1 Userfeedback-System

Abbildung 5.1 zeigt die verschiedenen Aktionen an, die ein normaler Benutzer und ein Administrator durchführen können. Nutzer sollen nur in der Lage Feedback zu geben, das dann durch das Backend in den Datastore gespeichert wird. Die verschiedenen Arten sind auf der Abbildung dargestellt und werden im Abschnitt 5.2 näher erläutert. Administrator haben zusätzlich die Möglichkeit, die Ergebnisse der Feedback-Komponenten in graphischer Form auf der Admin-Page einzusehen. Das soll den Administratoren ermöglichen, die Ergebnisse der Abfragen zu analysieren und mögliche Probleme zu beheben.

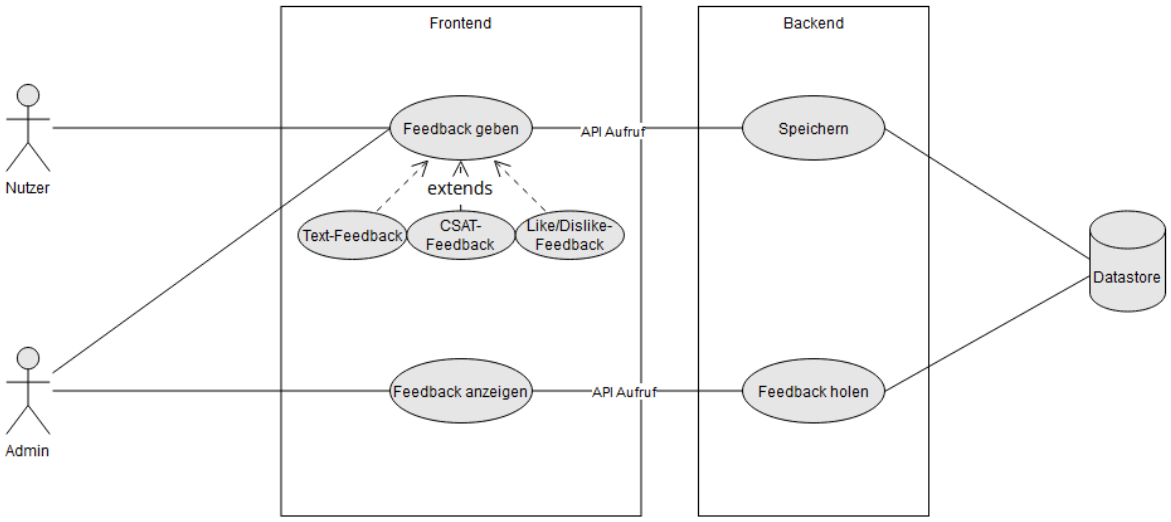

Abbildung 5.1 Mögliche Interaktionen für User und Administratoren

5.2 Datenmodell

In diesem Abschnitt werden die benötigten Daten für das Userfeedback-System anhand eines Datenmodells vorgestellt. Das Datenmodell wird in Form eines UML-Diagramms (siehe Abbildung 5.2) vorgestellt.

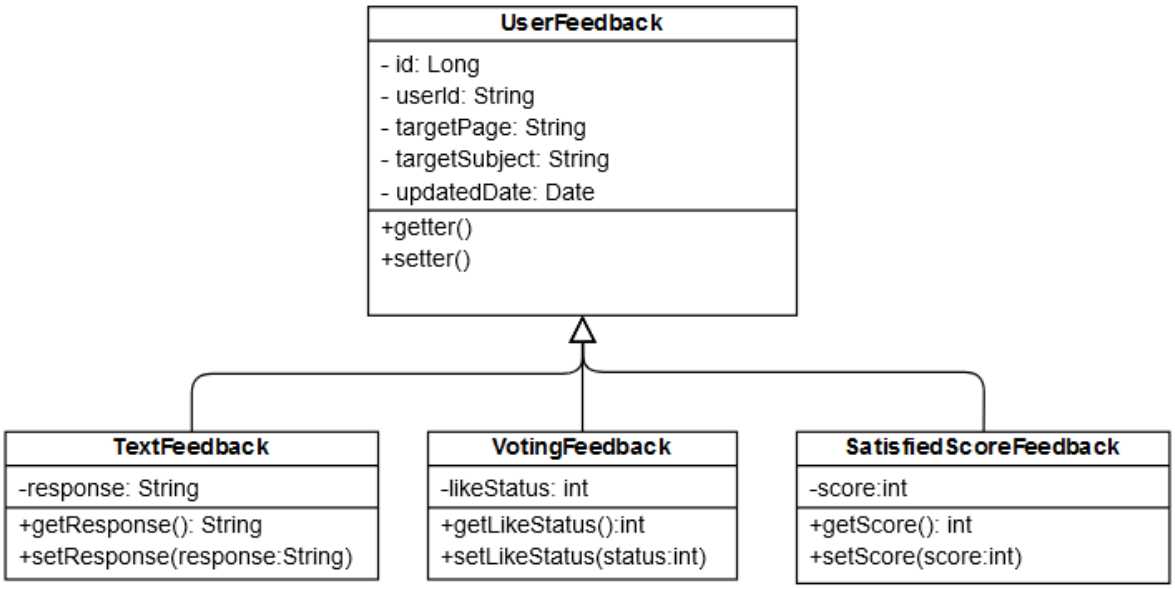

Abbildung 5.2 UML-Diagramm für die Feedbackarten

Die Klasse *UserFeedback* ist die Oberklasse aller Feedbackarten und beinhaltet Attribute, die in Tabelle 5.1 erläutert werden und die jede Feedbackart haben muss. Zu jedem Attribut gibt es eine entsprechende Setter- und Getter-Methode, mit denen die Werte geändert und geholt werden können.

Attribut	Typ	Bedeutung
id	Long	Eine automatisch erzeugte ID, mit der jedes Feedback eindeutig identifiziert werden kann
userId	String	Die ID des Benutzers, der das Feedback abgegeben hat
targetPage	String	Beschreibt für welche Seite das Feedback gegeben wurde und wird gespeichert mit den pathname[15] der URL. Beispiel: /de/personal-dashboard/
targetSubject	String	Soll beschreiben, für welche Komponente auf der Seite Feedback gegeben wurde. (Optional) Beispiel: CodingEditor
updatedDate	Date	Zeitpunkt der letzten Änderung/Anpassung

Tabelle 5.1 Attribute der UserFeedback Klasse

[15] https://developer.mozilla.org/en-US/docs/Web/API/Location/pathname

TextFeedback

Die Unterklasse *TextFeedback* speichert das qualitative Feedback, welches durch ein Textfeld gesammelt wurde. Sie beinhaltet neben den Attributen von UserFeedback ein spezifisches Attribut namens *response*, in dem der entsprechende Text gespeichert wird.

VotingFeedback

Die Unterklasse *VotingFeedback* speichert den Like-Status, der durch die Like/Dislike-Komponente gesammelt wurde. Sie beinhaltet neben den Attributen von *UserFeedback* ein spezifisches Attribut namens *likeStatus*, in das der Status in Form einer Zahl gespeichert wird. Der Status hat 3 Zustände, wie in der Tabelle 5.2 beschrieben.

Wert	Bedeutung
-1	Dem Benutzer gefällt der Inhalt nicht
0	Der Benutzer hat nicht abgestimmt
1	Dem Benutzer gefällt der Inhalt

Tabelle 5.2 Zustände für den likeStatus

SatisfiedScoreFeedback

Die Unterklasse *SatisfiedScoreFeedback* speichert den abgegebenen Wert, der durch die CSAT-Komponente gesammelt wurde. Sie beinhaltet als spezifisches Attribut den *score*, der in Form einer Zahl abgespeichert wird und die Werte 1 bis 5 annehmen kann, wobei 1 als „Nicht zufrieden“ und 5 als „Sehr zufrieden“ gedeutet wird.

5.3 Schichtenmodell von QDAcity im Backend

QDAcity nutzt im Backend das eine am häufigsten genutzte Strukturierungsprinzip für die Softwarearchitektur, das Schichtenmodell (Richards, 2015). In dieser wird die Anwendung in verschiedene Schichten unterteilt, wobei jede Schicht ihre eigenen spezifischen Aufgaben hat.

Der große Vorteil dieser Unterteilung ist die Trennung der Zuständigkeiten, denn jede Schicht hat ihre eigenen Aufgaben und kommuniziert mithilfe von Schnittstellen mit den angrenzenden Schichten. Dadurch kann jede Schicht getrennt voneinander getestet und gewartet werden, was die Qualität des Codes verbessert.

Das Backend-System von QDAcity kann in drei Schichten unterteilt werden, welche in Abbildung 5.3 mit Präsentationsschicht, Anwendungsschicht und einer Datenzugriffsschicht dargestellt sind. Die genaue Umsetzung dieses Modells wird in Abschnitt 6.2 näher beschreiben.

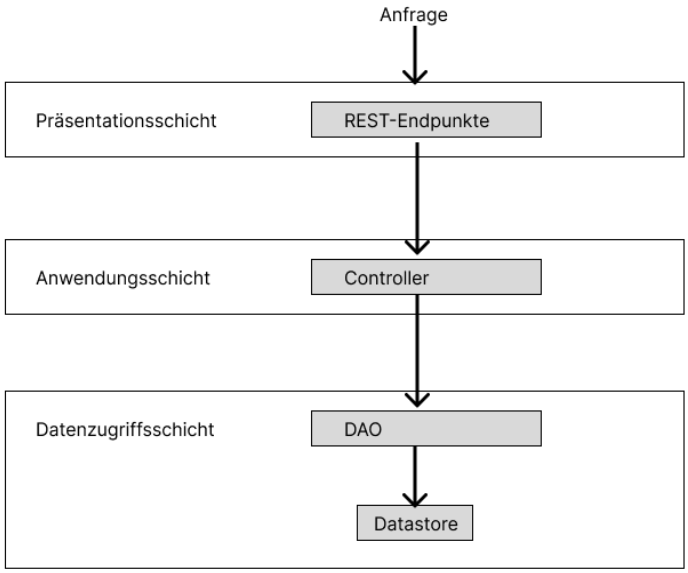

Abbildung 5.3 Schichtenmodell im Backend von QDAcity

Präsentationsschicht

Die Präsentationsschicht beinhaltet REST-Endpunkte, die überprüfen, ob der Nutzer autorisiert ist, Anfragen an den jeweiligen Endpunkt zu schicken. Die REST-Endpunkte werden vom Backend bereitgestellt und ermöglichen die Kommunikation zwischen Frontend und Backend mithilfe von HTTP-Anfragen. Diese Schicht beinhaltet keine Art von Logik und kann nur mit der Anwendungsschicht kommunizieren.

Anwendungsschicht

Die Anwendungsschicht spielt eine zentrale Rolle in dem Modell und stellt die Verbindung zwischen Präsentationsschicht und Datenzugriffsschicht her. Wie in vielen Anwendungen werden Controller-Klassen verwendet, um die Anfragen, die von der Präsentationsschicht kommen, zu koordinieren und zu steuern, und um die Businesslogik zu implementieren.

Datenzugriffsschicht

Die Datenzugriffsschicht ist für die Interaktionen mit der Datenbank zuständig. Sie benutzt DAO-Klassen (Data access object) in Kombination mit Objectify, um die CRUD-Operationen zu implementieren. Ist das Feedback erfolgreich eingefügt worden, wird ein Objekt mit den entsprechenden Daten und einer ID an die übergeordnete Schicht als Rückgabewert gegeben. Dieser Prozess geht bis zum Frontend zurück, damit die Komponente, die die Anfrage gesendet hat, eine passende Referenz hat, um das Feedback zu aktualisieren.

5.4 Frontend

In den vorhergehenden Abschnitten wurde ein Überblick über das Backend gegeben. Dieser Abschnitt befasst sich mit dem Ablauf der Komponenten im Frontend. Diese werden unterteilt in Feedback- und Admin-Komponenten. Für die Feedback-Komponenten wird anhand eines Sequenzdiagramms (siehe Abbildung 5.4) der allgemeine Ablauf erklärt, den jede Komponente durchlaufen soll.

5.4.1 Feedback-Komponenten

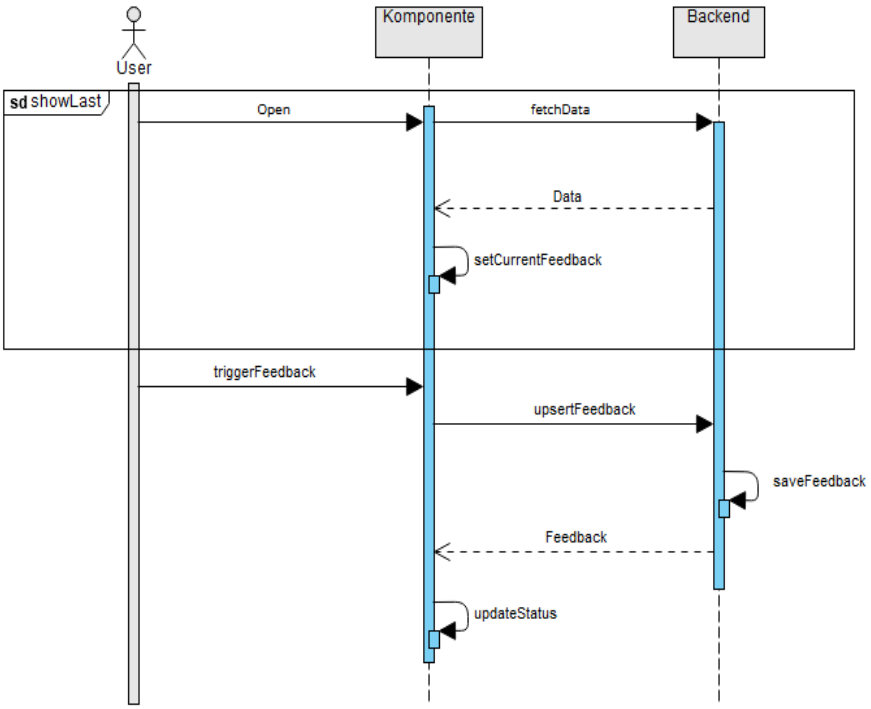

Abbildung 5.4 Sequenzdiagramm für den Ablauf einer Komponente

Beim Laden der Komponente werden, wenn nötig, die vorherigen Daten eines Feedbacks aus dem Backend geladen und der Zustand wird dementsprechend in der Komponente angepasst. Dieser Vorgang, der als Kasten mit Titel showLast gekennzeichnet ist, ist optional für Komponenten, die keinen Zustand haben. Er wird vor allem in der Like/Dislike-Komponente benötigt, da der angemeldete Nutzer in jeder Komponente nur einmal Feedback geben kann.

Gibt der Nutzer nun ein Feedback ab, wird in der Komponente ein Event ausgelöst, welches eine Backend *upsert* API-Methode ausführt. *Upsert* ist ein Sammelbegriff für die *update*- und *insert*-Operation und ist für das Aktualisieren und Einfügen von Entitäten ins Backend verantwortlich. Nach dem erfolgreichen Speichern der Feedback-Entität im Backend wird ein Objekt mit den entsprechenden Daten und ID zurückgegeben. Der Zustand der Komponente wird dann der Eingabe konform aktualisiert.

5.4.2 Admin-Komponenten

Bei den Admin-Komponenten gibt es eine weitere Unterscheidung in allgemeine Statistiken und Feedbackart spezifisch Diagramme.

Allgemeine Statistiken

Diese Art von Komponenten muss eigenständig beim Laden der Seite die Werte mit Hilfe einer passenden REST-Endpunkt Anfrage laden und anzeigen. Dabei muss jede Anfrage die drei Schichten, aus Abschnitt 5.3, durchlaufen. Mit dem Ergebnis der Anfrage wird die Komponente dann mit dem richtigen Wert dargestellt.

Spezifische Feedback Diagramme

Für die Implementierung der Diagramme sollen drei Dropdownmenüs für Zeitraum, *targetPage* und *targetSubject* implementiert werden. Diese Dropdownmenüs sollen eigenständig Feedbackart spezifische *targetPages* und *targetSubjects* als Auswahloptionen anbieten. Sobald die Auswahl an einer der drei Dropdowns geändert wird, soll die Komponente die Daten im Diagramm mit einem Endpunkt-Methodenaufruf aktualisieren.

6 Design & Implementierung

Im Folgenden wird die Implementierung der Schichten im Backend vorgestellt. Daraufhin folgen das Design und die Implementierung der Komponenten im Frontend. Sowohl die Frontend als auch die Backend Implementierungen folgen der Architektur aus dem vorhergehenden Kapitel, und sollen die Anforderungen aus Kapitel 4 erfüllen. Die Umsetzung erfolgt im Backend mithilfe von Java 8 und im Frontend mithilfe von JavaScript und der JavaScript-Programmbibliothek React. Für das Design der Komponenten wurden die bereits von QDAcity benutzten Komponenten wie *PrimaryButton*[16], welcher ein Button Element in QDAcity Design beschreibt, sowie die Icons von der Icon Bibliothek Font Awesome[17] genutzt. Für die Darstellung der Diagramme werden Google Charts[18] verwendet. Am Ende des Kapitels werden die Schritte beschrieben, die erfolgen müssen, um die Komponente wiederzuverwenden und um das System um eine weitere Komponente zu erweitern.

6.1 Backend

6.1.1 UserFeedbackEndpoint

Die Klasse *UserFeedbackEndpoint* repräsentiert die Präsentationsschicht und ist für die Autorisierung des Nutzers und die Weiterleitung der Anfrage zu der Controllerklasse zuständig. Es wurden insgesamt 16 REST-Endpunkte für das Userfeedback-System erstellt, die vom Frontend aufgerufen werden können. Alle Endpunkte können im Anhang eingesehen werden.

```
1  @ApiMethod(
2          name = "userFeedback.deleteFeedback",
3          httpMethod = "DELETE",
4          path = "userFeedbacks/{ID}")
5  public void deleteFeedback(@Named("ID") Long ID, User user)
6          throws UnauthorizedException{
7          throwIfUnauthorized(user);
8          Context.executeWith(user, context -> {
9                  UserFeedbackController.with(context).deleteFeedback(ID);
10         });
11 }
```

Codebeispiel 6.1 Beispiel für eine Endpunkt-Methode

In dem Codebeispiel 6.1 wird durch die Annotation *@ApiMethod* (Zeile 1) kenntlich gemacht, dass es sich bei der *deleteFeedback*-Methode um eine Endpunkt-Methode handelt. In dieser wird mithilfe der *throwIfUnauthorized*-Methode (Zeile 7) überprüft, ob es sich um einen angemeldeten/autorisierten Benutzer handelt. Sollte das nicht der Fall sein, wirft die Methode eine *UnauthorizedException* und die Anfrage wird nicht an den Controller weitergegeben. Gelingt die Überprüfung, wird die Anfrage an die *UserFeedbackController*-Klasse ohne den User Parameter weitergeleitet, weil dieser in diesem Schritt schon autorisiert wurde. Context.*executeWith* (Zeile 8) sorgt dafür, dass der Aufruf in den darauffolgenden Schichten mit immer demselben autorisierten User erfolgt. Zudem kontrolliert sie den PersistenceManager[19], der für die CRUD-Operation im Backend zuständig ist.

[16] https://gitlab.com/profoss/qdacity/qdacity/-/wikis/Buttons-&-links#basic-buttonlinks-buttons-and-icons
[17] https://fontawesome.com/
[18] https://developers.google.com/chart/interactive/docs/gallery?hl=de
[19] https://docs.oracle.com/cd/E13189_01/kodo/docs341/jdo-javadoc/javax/jdo/PersistenceManager.html

6.1.2 UserFeedbackController & UserFeedbackDAO

Die Anwendungsschicht wird durch die *UserFeedbackController* Klasse realisiert. In der aktuellen Implementierung ist die Klasse für das Erstellen und Weiterleiten der Entitäten zum *UserFeedbackDAO* zuständig. Die *UserFeedbackDAO* Klasse implementiert die Datenzugriffschicht und hat die Aufgabe, alle CRUD-Operationen durchzuführen und die eingefügte Entität der übergeordneten Anwendungsschicht zurückzugeben.

6.2 Like/Dislike-Komponente

6.2.1 Design

Die Like/Dislike-Komponente besteht aus zwei Button Elementen, die für „Gefällt mir“ und „Gefällt mir nicht“ stehen, und einem Text Element. Das Text Element soll die Zugehörigkeit der Komponente beschreiben und beinhaltet eine Frage. Die aktuelle Auswahl soll farblich, durch die Primärfarbe Blau von QDAcity dargestellt werden.

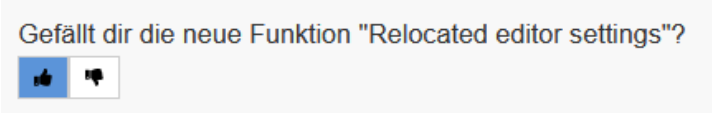

Abbildung 6.1 Design für Like/Dislike-Komponente

6.2.2 Implementierung

Nach der Initialisierung der Komponente wird mit dem *useEffect* Hook von React[20] und einem leeren dependencies Array[21] (siehe Codebeispiel 6.2 Zeile 2) einmalig geprüft, ob der angemeldete Benutzer schon einen Like/Dislike für dieses bestimmte Ziel abgegeben hat. Das Ergebnis dieses Aufrufes wird dann in einem *useState* Hook[22] gespeichert. Der *useState* Hook gibt einen zustandsbezogenen Wert und eine Funktion zurück, mit der diese geändert werden können (siehe Codebeispiel 6.2 Zeile 1). Zusätzlich sollen nicht angemeldete Nutzer auf der Public-Page Feedback geben können. In dem Fall wird beim Laden der Seite das initiale Holen eines möglich vorhandenen Feedbacks mit einer if-Überprüfung (siehe Codebeispiel 6.2 Zeile 3) übersprungen, weil es nicht möglich ist, einem nicht eingeloggten User eine ID zuzuordnen.

Klickt der Nutzer auf einen der beiden Buttons, wird eine *upsert*-Methode aufgerufen, welche bei nicht vorhandenem Feedback ein neues Feedback speichert und bei bereits Vorhandenem dieses dementsprechend aktualisiert. Anschließend wird der ausgewählte Button mit CSS-Stylings farblich durch die Primärfarbe Blau von QDAcity angepasst.

```
1  const [feedback, setFeedback] = useState({});
2  useEffect(() => {
3   if (loggedIn) {
4          const getInitialLikeStatusAndFeedback = async () => {
5                  const result = await getFeedback(targetSubject, targetPage);
6                  setFeedback(result);
7                  };
8          getInitialLikeStatusAndFeedback().catch(console.error);
9   }
10 }, []);
```

Codebeispiel 6.2 useState und useEffect in der Like/Disklike-Komponente

[20] https://react.dev/reference/react/useEffect
[21] https://react.dev/reference/react/useEffect#examples-dependencies
[22] https://react.dev/reference/react/useState

6.3 Customer Satisfaction Score & Text-Komponente

6.3.1 Design

Die CSAT & Text-Komponente durchläuft insgesamt drei verschiedene Phasen.

Die erste Phase wird in Abbildung 6.2 dargestellt und besteht aus einem Button, der unten rechts im Coding-Editor platziert ist. Die Platzierung ist so gewählt, dass sie den Benutzer nicht bei der Arbeit stört, aber dennoch einen einfachen Weg bieten soll, Feedback zu geben.

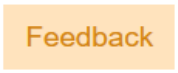

Abbildung 6.2 Erste Phase der CSAT & Text-Komponente

Entscheidet sich der Benutzer Feedback zu geben, und klickt auf den Button, erscheint ein kleines Fenster, in dem Rückmeldung gegeben werden kann (siehe Abbildung 6.3). Das Fenster besteht aus einem Senden-Button, einem Text-Element, einer CSAT-Komponente, welche durch fünf Emojis repräsentiert wird, und einem Freitext-Element, in dem die Auswahl näher beschrieben werden kann. Zusätzlich gibt es die Option, mithilfe einer Checkbox die Komponente auszublenden, falls der Benutzer kein Feedback geben will.

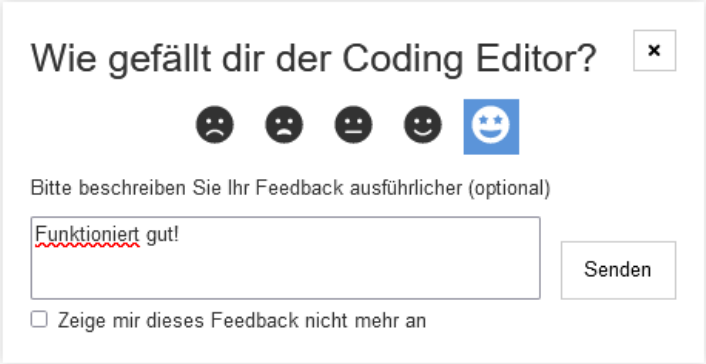

Abbildung 6.3 Zweite Phase der CSAT & Text-Komponente

Sobald ein CSAT- oder Text-Feedback angegeben wurde, wird mit dem Klick auf den Senden-Button die letzte Phase, die in Abbildung 6.4 dargestellt ist, der Komponente eingeleitet und das Fenster wird durch ein anderes ersetzt, um den Nutzer über das erfolgreiche Senden zu informieren. Ist kein Feedback angegeben, passiert nichts beim Klicken des Senden-Buttons. Dieses verschwindet nach drei Sekunden. Anschließend wird, je nach Auswahl der Checkbox, wird Phase eins der Komponente wieder angezeigt oder nicht.

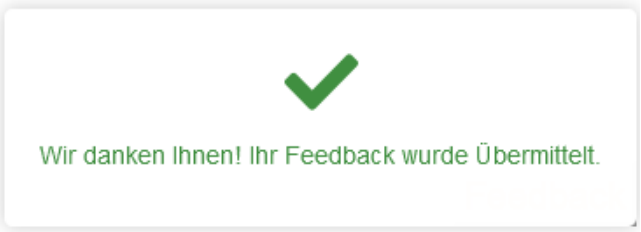

Abbildung 6.4 Letzte Phase der CSAT & Text-Komponente

6.3.2 Implementierung

Wie im vorherigen Abschnitt 6.1.2 beschrieben, durchläuft die CSAT & Texte-Komponente drei Phasen, die mit zwei booleschen States, *showDialog* und *showThankYou* realisiert werden, wobei diese anfangs mit false initialisiert werden. Drückt der Benutzer nun auf den orangenen Button (siehe Abbildung 6.2), wird der State *showDialog* zu true. Durch ein Conditional Rendering[23] erscheint nun das Hauptfenster der Komponente (siehe Abbildung 6.3). Die Auswahl der verschiedenen Teilkomponenten (Freitext und CSAT) wird mit CSS-Styling angepasst und beim Klick des Senden-Button werden mithilfe von den Endpunkt-Methoden *insertTextFeedback* und *insertSatisfiedFeedback* die jeweiligen Feedbacks abgespeichert. Für das Speichern der Checkboxauswahl wurde der Local Storage[24] verwendet, um die Anfragen an das Backend zu minimieren.
Nachdem alles erfolgreich gesendet wurde, wird der State *showDialog* zu false geändert, um das Hauptfenster zu schließen, und der State *showThankYou* wird zu true. Durch die Änderung wird nun das Danke-Fenster angezeigt. Nach einem Timeout von drei Sekunden wird der Wert wieder auf false gesetzt, was wiederum zum Ursprungszustand zurückführt. Hat der Benutzer die Checkbox ausgewählt, mit der das Feedback ausgeblendet werden kann, wird der Anfangsbutton mit Conditional Rendering ausgeblendet. Bei jedem Laden der Seite wird nun überprüft, ob es einen *hideFeedbackButton* Key im Local Storage gibt. Der Button wird dementsprechend gerendert oder nicht. Die Einstellung im Local Storage bleibt 30 Tage oder bis der Eintrag im Local Storage gelöscht wird bestehen.

[23] https://react.dev/learn/conditional-rendering
[24] https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage?retiredLocale=de

6.4 Admin-Page-Komponenten

Die eingesammelten Feedbacks sollen auf der Admin-Page graphisch dargestellt werden, um dem Administrator einen einfachen Weg zu geben, das Feedback zu analysieren. Diese werden unterteilt in allgemeine Feedbackstatistiken und für Feedbackart spezifische Tabellen und Graphen.

6.4.1 Allgemeine Statistiken

6.4.1.1 Design

Analog zu der bisherigen Admin-Page, vorgestellt in Abschnitt 2.1.4, sollen allgemeine Statistiken wie gesamte Likes/Dislikes und CSAT-Score oben auf der Seite dargestellt werden (siehe Abbildung 6.5). Das hat den Vorteil, dass man auf dem ersten Blick erkennen kann, wie gut die Gesamtanwendung bei den Benutzern ankommt.

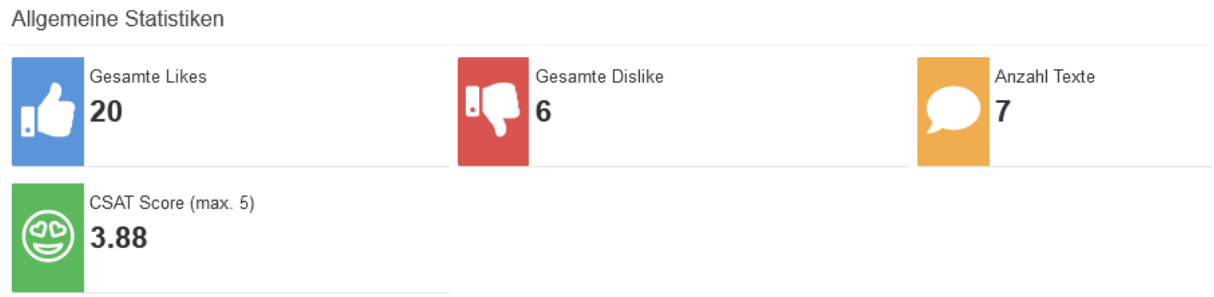

Abbildung 6.5 Design für allgemeine Feedbackstatistiken

6.4.1.2 Implementierung

Ähnlich wie bei der Like/Dislike-Komponente benutzt diese Komponente auch den *useEffect* Hook in Kombination eines leeren dependencies Arrays, um die Werte der verschiedenen Komponente initial zu laden. Anschließend wird jeder einzelne Wert mit einem Icon und Farbe an eine von QDAcity bereitgestellten *ValueBox* übergeben.

6.4.2 FeedbackTargetChooser-Komponente

6.4.2.1 Design

Um die Daten der Diagramme zu ändern, sollen Dropdownmenüs implementiert werden, die je nach Feedbackart eine Liste aller *targetPages* bzw. *targetSubject* bereitstellt, mit denen nach bestimmten Kriterien gefiltert werden kann. Das Design orientiert sich an dem bereits von QDAcity implementieren Zeitraumfilter (*TimeFrameFilter*) und wird in Abbildung 6.6 gezeigt.

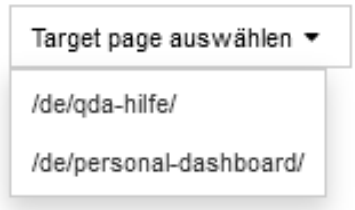

Abbildung 6.6 Design für FeedbackTargetChooser-Komponente

6.4.2.2 Implementierung

Für die Implementierung des *FeedbackTargetChooser* wurde der *DropDownButton*, der bereits im *TimeFrameFilter* benutzt wurde, wiederverwendet. Dieser *DropDownButton* sorgt eigenständig für die Dropdownstruktur und benötigt eine Liste an Items, die angezeigt werden sollen. Bei der Auswahl von einem der Items wird eine Funktion aufgerufen, die von der Komponente, die den *FeedbackTargetChooser* benutzt, abgefangen werden kann, die dann ihrerseits eine Folgefunktion aufrufen kann.

```
1  <FeedbackTargetChooser
2  onChangeTarget={(target) => {
3          setTargetPage(target);
4  }}
5  targetPageList={targetPageList}
6          initText={'Choose target page'}
7  />
8  <FeedbackTargetChooser
9          onChangeTarget={(target) => {
10         setSelectedSubject(target);
11         }}
12         targetPageList={targetSubjectList}
13         initText={'Choose subject'}
14 />
```

Codebeispiel 6.3 Verwendung des FeedbackTargetChooser in der LikeDislikeChart

In Codebeispiel 6.3 wird die Verwendung der zwei *FeedbackTargetChoosers* im *LikeDislikeChart* dargestellt. Hier wird eine *targetPageList* als Props[25] übergeben, die eine gültige Liste von *TargetPages* beinhaltet. Gleichzeitig wird der Komponente eine Funktion *onChangeTarget* mit einem *target* Parameter übergeben. Ruft die *FeedbackTargetChooser*-Komponente nun die übergebene Funktion auf, wird in der *LikeDislikeChart* die Funktion *setTargetPage* (Zeile 3) mit dem entsprechenden Parameter *target* aufgerufen.
Die *setTargetPage*-Funktion speichert die Auswahl in einer State-Variable und sucht mit einer Endpunkt-Methode jedes eindeutige *targetSubject* zu einer bestimmten *targetPage*. Die gefundenen Einträge werden dann als *targetSubjectList* (Zeile 12) in den nächsten *FeedbackTargetChooser* übergeben.
Mit diesem Prinzip ist es möglich, immer die entsprechende *targetSubjectList* zu einer bestimmten *targetPage* anzuzeigen.

[25] https://legacy.reactjs.org/docs/components-and-props.html

6.4.3 LikeDislikeChart-Komponente

6.4.3.1 Design

Als Darstellungsform für die Like/Dislike-Komponente wurde ein Balkendiagramm gewählt. In diesem lässt sich das Verhältnis von Likes und Dislikes in einem bestimmten Zeitraum gut erkennen und darstellen (siehe Abbildung 6.7). Das Filtern nach Datum, *targetPage* und *targetSubject* wird mithilfe eines *TimeFrameFilters* und zwei *FeedbackTargetChoosers* realisiert.

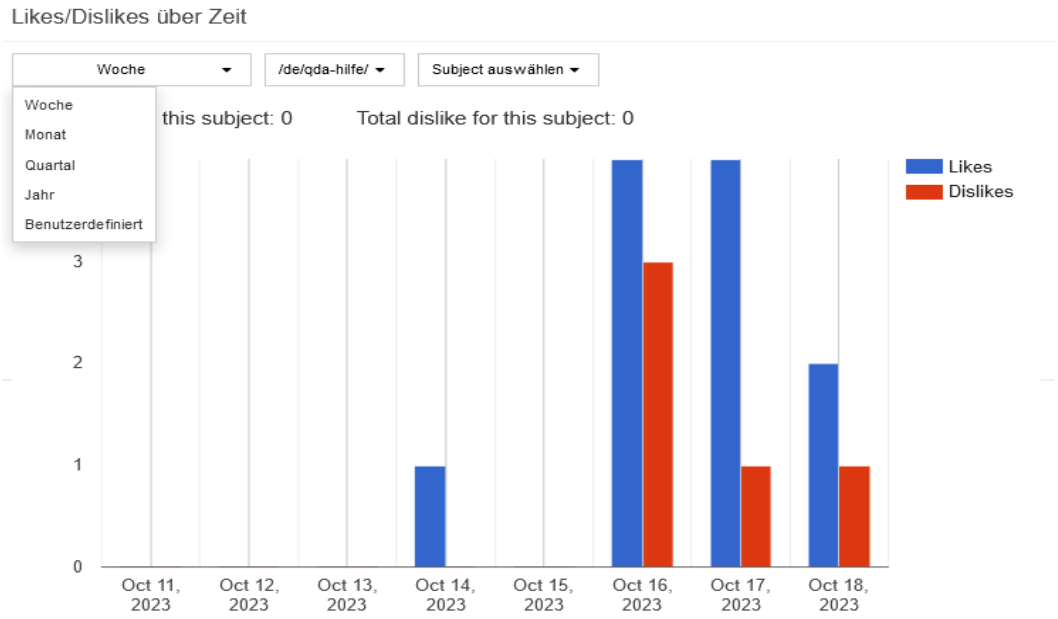

Abbildung 6.7 Diagramm für Like/Dislike-Komponente

6.4.3.2 Implementierung

Die *LikeDislikeChart*-Komponente nutzt die übergebenen Daten der drei Dropdownmenüs und sucht mit der Endpunkt-Methode *listUserFeedbacks* alle Feedbacks, die zu der Auswahl passen. Um sicherzustellen, dass die richtigen Feedbacks geholt werden, werden fünf Parameter (siehe Tabelle 6.1) der *listUserFeedbacks*-Methode übergeben.

Name	Datentyp	Nutzen
targetPage	String	Seite für die Feedback geholt werden soll
targetSubject	String	Komponente auf der Seite für die Feedback geholt werden soll
userFeedbackType	String	Die Feedbackart, die geholt werden soll.
startDate	Date	Anfangsdatum des Zeitraumes
endDate	Date	Enddatum des Zeitraumes

Tabelle 6.1 Parameter der listUserFeedbacks Endpunkt-Methode

Mit dem *useEffect* Hook und einem dependecies Array mit *startDate*, *endDate*, *targetPage* und *targetSubject* wird nun bei jeder Änderung an einem der Dropdownmenüs die Endpunkt-Methode neu aufgerufen. Somit bekommt das benutzte Säulendiagramm von Google Charts[26] immer die aktuellen Daten, die zur Auswahl der Dropdownmenüs passen.

[26] https://developers.google.com/chart/interactive/docs/gallery/columnchart?hl=de#overview

6.4.4 TextTable-Komponente

6.4.4.1 Design

Um die Text-Feedbacks darzustellen, wurde eine Tabelle verwendet, da anhand des Tabellenheader gut erkennbar ist, wer, wann und wofür ein Nutzer Feedback gegeben hat. Zusätzlich sollen drei Dropdownmenüs ermöglichen, die Tabelle nach verschiedenen Kriterien zu filtern (Siehe Abbildung 6.8).

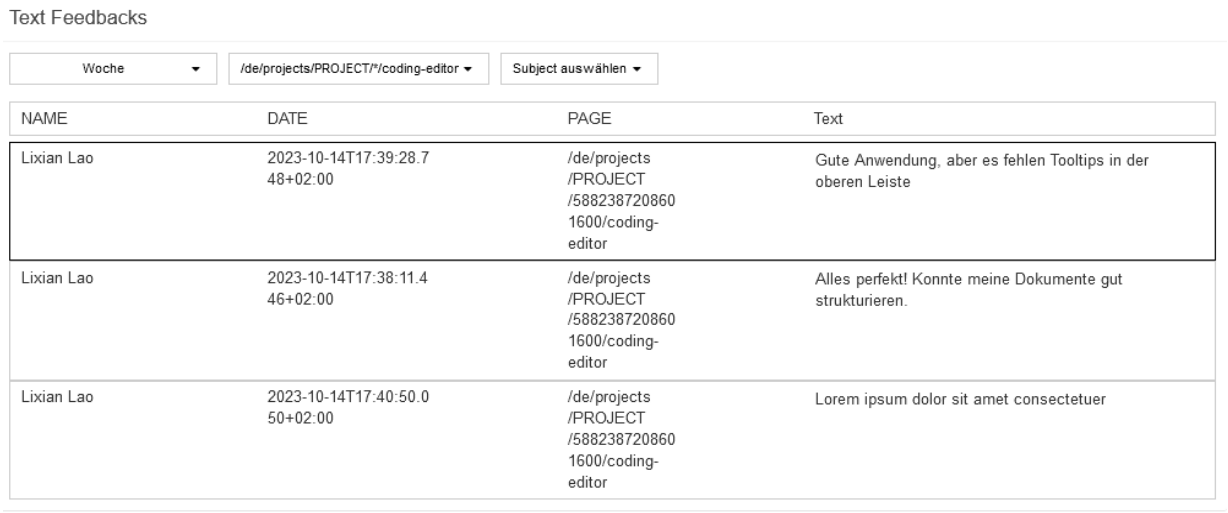

Abbildung 6.8 Diagramm für Text-Feedback

6.4.4.2 Implementierung

Die *TextTable*-Komponente funktioniert genau wie die *LikeDislikeChart*-Komponente. Sie aktualisiert bei jeder Änderung an den Dropdownmenüs die Daten, die an die von QDAcity bereitgestellten *ItemList*-Komponente übergeben werden. Sie besitzt eine eingebaute Seitennummerierung, um die Darstellung der Tabelle kompakt zu halten. Um sich an die Größe anderer Admin-Komponenten anzupassen, werden fünf Einträge pro Seite angezeigt.

6.5 Wiederverwendung der Komponenten

Jede Frontend Komponente, die im Abschnitt 6.3 vorgestellt wurde, sorgt eigenständig für die richtigen Endpunkt-Methoden Aufrufe. Für die Wiederverwendung der Komponente muss diese im ersten Schritt importiert werden. Dann kann die Komponente wie ein HTML-Element benutzt werden. Wichtig dabei ist, dass der Komponente die richtigen Props übergeben werden. Diese sind in der Implementierung der Komponente definiert und sind gekennzeichnet mit *propTypes* (siehe Codebeispiel 6.4, Zeile 20-22).

```
1  import React from 'react';
2  import PropTypes from "prop-types";
3  import FeedbackTargetChooser from "./FeedbackTargetChooser";
4
5
6  const TestChart = ({ targetPageList }) => {
7  return (
8  <div>
9  <FeedbackTargetChooser
10 onChangeTarget={(target) => {
11 //onChangeFunction
12 }}
13 targetPageList={targetPageList}
14 initText={'Choose target page'}
15 />
16 </div>
17 );
18 };
19
20 TestChart.propTypes = {
21 targetPageList: PropTypes.array.isRequired,
22 };
23
24 export default TestChart;
```

Codebeispiel 6.4 Wiederverwendung der FeedbackTargetChooser-Komponente

Im Codebeispiel 6.4 wird die Einbindung des *FeedbackTargetChooser* in einer *TestChart*-Komponente gezeigt. Die *TestChart* umhüllt die *FeedbackTargetChooser*-Komponente in ein -Element und rendert die Elemente. *FeedbackTargetChooser* hat in diesem Fall drei Props:

onChangeTarget: Eine Callback-Funktion, die eine Funktion von *TestChart* aufrufen soll, wenn sich die Auswahl des Dropdowns ändert.

targetPageList: Eine Liste von *targetPages*, die die *FeedbackTargetChooser* anzeigen soll.

initText: Der Initialtext, der angezeigt werden soll, wenn noch nichts ausgewählt wurde.

Die Wiederverwendung der *TestChart*-Komponente funktioniert ähnlich. Der Unterschied ist, dass sie nur ein Prop braucht, welches in Codebeispiel 6.4, Zeile 21 definiert wird.

6.6 Erweiterung des Userfeedback-Systems

Dieser Abschnitt befasst sich mit den Schritten, die eingehalten werden müssen, um das System um eine Komponente zu erweitern.

Backend

Im Backend muss im ersten Schritt eine neue Unterklasse von *UserFeedback* erstellt werden, die in die Datenbank gespeichert wird. Neben der Implementierung der passenden Methoden in den Schichten, die im Abschnitt 5.3 vorgestellt wurden, muss die *UserFeedbackType.java* um die neue Unterklasse erweitert werden. Der *UserFeedbackType* wird benutzt, um nach einer bestimmten Feedbackart zu filtern. Die Filterung nach einer bestimmten Art von Feedback wird vor allem in den Admin-Komponenten verwendet.

Frontend

Im Frontend müssen zwei Komponenten implementiert werden. Einmal die Komponente zum Sammeln von Feedback und einmal die Komponente auf der Admin-Page.
Für die Komponente auf der Admin-Page können die Endpunkt-Methode *listUserFeedback* und die *FeedbackTargetChooser*-Komponente wiederverwendet werden, um Daten aus dem Backend zu laden.

7 Evaluation

Dieses Kapitel befasst sich damit, ob die funktionalen und nicht funktionalen Anforderungen aus Kapitel 4 erfüllt worden sind.

7.1 Funktionale Anforderungen

7.1.1 Like/Dislike-Komponente

FA 1: Die Like/Dislike-Komponente muss die Zugehörigkeit des Feedbacks anzeigen.

Es wird bei jeder Komponente ein Text angezeigt, der für die Zugehörigkeit verantwortlich ist (siehe Abbildung 7.1).

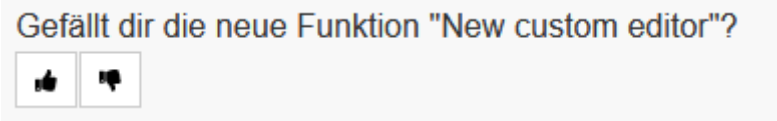

Abbildung 7.1 Beispiel für die Like/Dislike-Komponente

Funktionale Anforderung wurde erfüllt.

FA 2: Die Like/Dislike-Komponente muss die Möglichkeit bieten, einen Button zu klicken.

In der Anwendung werden die HTML <button>-Elemente benutzt, die ein *OnClick* Event abfangen und den Zustand der Komponente sowohl im Code als auch graphisch dementsprechend aktualisieren.

Funktionale Anforderung wurde erfüllt.

FA 3: Die Like/Dislike-Komponente sollte 3 Zustände haben (Liked/Disliked/NotSelected).

Da es zwei Buttons gibt und nur höchstens einer aktiv sein soll, gibt es insgesamt drei verschiedene Zustände, die die Komponente annehmen kann.

Funktionale Anforderung wurde erfüllt.

FA 4: Die Like/Dislike-Komponente muss fähig sein, den aktuellen Zustand farbig zu repräsentieren.

Die Buttons werden je nach Like-Zustand mit einem CSS-Styling versehen.

Funktionale Anforderung wurde erfüllt.

FA 5: Die Like/Dislike-Komponente sollte fähig sein, den aktuellen Zustand nach einem neu laden der Seite korrekt anzuzeigen.

Beim Laden der Komponente wird durch einen REST-API-Aufruf nach schon existierendem Feedback geprüft. Dementsprechend wird der Zustand der Komponente sowohl im Code als auch graphisch aktualisiert.

Funktionale Anforderung wurde erfüllt.

7.1.2 Customer Satisfaction Score & Text-Komponente

FA 6: Die CSAT & Text-Komponente muss die Möglichkeit bieten, Feedback mit einem Button klick zu senden.

Durch einen Klick auf den Senden Button in der zweiten Phase der Komponente werden die Ergebnisse der beiden Teilkomponenten CSAT und Text als Anfrage an REST-Endpunkten geschickt und im *UserFeedbackDAO* in den Datastore gespeichert.

Funktionale Anforderung wurde erfüllt.

FA 7: Die CSAT & Text-Komponente muss die Zugehörigkeit des Feedbacks anzeigen.

Es wird bei jeder Komponente ein Text angezeigt, der für die Zugehörigkeit verantwortlich ist.

Funktionale Anforderung wurde erfüllt.

FA 8: Die CSAT & Text-Komponente muss eine Rückmeldung geben, wenn ein Feedback versendet worden ist.

Die letzte Phase der Komponente dient als Rückmeldung und wird nur angezeigt, wenn das Senden eines Feedbacks erfolgreich war.

Funktionale Anforderung wurde erfüllt.

FA 9: Die CSAT & Text-Komponente soll dem Benutzer die Möglichkeit geben, die Komponente auszublenden.

In der CSAT & Text-Komponente wurde eine Möglichkeit hinzugefügt mit einer Checkboxauswahl die Komponente dauerhaft auszublenden. Diese Änderung wird im Local Storage gespeichert und hält 30 Tage oder bis der Cache des Browsers gelöscht wird.

Funktionale Anforderung wurde erfüllt.

7.1.3 Admin-Page-Komponenten

FA 10: Die Admin-Page-Komponenten müssen die richtigen Feedbacks aus dem Backend holen.

Durch den *useEffect* Hook wird sichergestellt, dass sich bei jeder Änderung der Dropdownmenüs die Diagramme aktualisieren. Somit ist garantiert, dass das Diagramm immer die Daten passend zu der Auswahl anzeigt.

Funktionale Anforderung wurde erfüllt.

FA 11: Die Admin-Page-Komponenten müssen dem Administrator ermöglichen, zwischen verschiedenen Feedbackarten zu wechseln.

Der ursprüngliche Plan war es, nur eine Tabelle/Graphen zu benutzen, um die Ergebnisse anzuzeigen und die Feedbackarten mit einem Dropdownmenü oder Ähnlichem zu wechseln. Diese Variante hat sich als nicht sinnvoll erwiesen, weil ein Amin möglicherweise mehrere Feedbackarten gleichzeitig analysieren will und wurde deswegen aufgeteilt in eine Graphik pro Feedbackart.

Funktionale Anforderung wurde nicht erfüllt.

FA 12: Die Admin-Page-Komponenten müssen dem Administrator eine graphische Darstellung passend zum Feedbacktyp zeigen.

Im Abschnitt 6.1.3 wurden die Vorteile der Auswahl der Diagramme besprochen.

Funktionale Anforderung wurde erfüllt.

FA 13: Die Admin-Page-Komponenten müssen dem Administrator allgemeine Statistiken, wie Ge-
samte Like/Dislike oder CSAT-Score anzeigen.

Für jede Feedbackkomponente wurde, wie in 6.1.3 gezeigt eine „allgemeine Statistik"-Komponente erstellt.

Funktionale Anforderung wurde erfüllt.

7.2 Nicht-Funktionale Anforderungen

7.2.1 Funktionalität

NFA 1: Das System soll keine existierenden Funktionen von QDAcity einschränken oder beeinträchtigen.

Da während dem ganzen Entwicklungsprozess Unit-Tests und Code-Reviews durchgeführt wurden, konnte bei jedem Schritt getestet werden, ob die eingefügte Implementierung das existierende System beeinträchtigt. Es wurden keine Funktionalitäten eingeschränkt.

Nicht-Funktionale Anforderung wurde erfüllt.

7.2.2 Kompatibilität

NFA 2: Das System muss mit den verwendeten Diensten (Google App Engine, Datastore, Cloud Endpoints) kompatibel sein.

Bei der Implementierung des ganzen Systems wurde darauf geachtet, dass die gleichen Dienste, wie Google App Engine und Datastore mit Objectify genutzt werden. Es wurden keine weiteren Dienste in das System eingefügt.

Nicht-Funktionale Anforderung wurde erfüllt.

7.2.3 Benutzbarkeit

NFA 3: Das System ist an das Farbschema von QDAcity angepasst.

Alle Farbtöne, die für die CSS-Stylings genommen wurden, kommen aus der *Theme.js*, die alle Farbschemen von QDAcity definiert.

Nicht-Funktionale Anforderung wurde erfüllt.

NFA 4: Das System soll Tastaturbedienung unterstützen.

Alle HTML-Elemente wie <button> und <textarea>, die im System benutzt wurden, haben eine eingebaute Tastaturunterstützung, die mit Hilfe von Semantischer HTML funktioniert[27].

Nicht-Funktionale Anforderung wurde erfüllt.

[27] https://developer.mozilla.org/en-US/docs/Learn/Accessibility/HTML

7.2.4 Sicherheit

NFA 5: Das System muss bei jedem Endpunkt-Aufruf überprüfen, ob der Anwender autorisiert ist, diese zu benutzen.

Alle REST-Endpunkte des Userfeedback Systems prüfen eigenständig, ob der Benutzer eingeloggt und autorisiert ist, diesen Endpunkt aufzurufen. Im Fehlerfall wird eine *UnautorizedException* geworfen und die Anfrage wird nicht weitergeleitet.

Nicht-Funktionale Anforderung wurde erfüllt.

7.2.5 Wartbarkeit

NFA 6: Das System ist mit entsprechenden Tests abgedeckt (Unittests, Akzeptanztest Test).

Die Funktionalitäten im Backend wurden mit Unittests überprüft. Der Fokus lag auf der Code Coverage und der Implementierung der Businesslogik. Es wurden insgesamt 31 Tests im Backend erstellt, die sowohl Fehlerfälle als auch Edge-Cases getestet haben. In der Tabelle 7.1 wird die Methoden und Testabdeckung für die verschiedenen Schichten und Klassen dargestellt.

Klasse	Methode, %	Line, %
UserFeedbackEndpoint	100% (33/33)	100% (57/57)
UserFeedbackController	100% (20/20)	93% (107/114)
UserFeedbackDAO	100% (10/10)	100% (24/24)
UserFeedback	100% (10/10)	100% (11/11)

Tabelle 7.1 Testabdeckung fürs Backend

Zusätzlich zu den Unit Tests waren Akzeptanztests geplant, aber diese konnten aus zeitlichen Gründen nicht umgesetzt werden.

Nicht-Funktionale Anforderung nur teilweise erfüllt.

NFA 7: Die Komponenten sind modular und können wiederverwendetet werden.

Bei der Entwicklung jeder Komponente wurde darauf geachtet, dass sie so modular wie möglich implementiert wird. Das bedeutet jede Komponente kann eigenständig funktionieren. Zusammen mit dem Prinzip der Components von React, kann jede Komponente wiederverwendet werden, ohne dabei Funktionalität zu verlieren.

Nicht-Funktionale Anforderung wurde erfüllt.

8 Ausblick

Mit dem Einfügen der in Kapitel 6 beschriebenen Komponenten hat QDAcity den ersten großen Schritt in Richtung kontinuierlicher Verbesserung gemacht. Dennoch gibt es weitere Feedbackarten, die das System weiter verbessern können. Das folgende Kapitel gibt einen kurzen Überblick über mögliche Verbesserungen.

Willingness-To-Pay

Angesicht der baldigen Einführung eines Abonnements für QDAcity würde sich anbieten, das Userfeedback-System um eine WTP-Komponente zu erweitern. Diese wurde im Abschnitt 3.2 vorgestellt und sammelt Daten in Form von Umfragen. Als Ort für die Komponente wäre die Preisseite von QDAcity[28] oder eine E-Mail-Umfrage sinnvoll. Als Darstellungsform wird normalerweise ein Average-Min-Max-Diagramm benutzt, aber Google Charts bietet aktuell keine Implementierung für diesen Diagrammtyp an. Die Alternative, die benutzt werden kann, ist das Kombinationsdiagramm[29], um Minimum, Maximum und Durchschnitt anzuzeigen. In Abbildung 8.1 wird ein Beispiel eines solchen Diagramms gezeigt. In diesem werden die Werte erkenntlich dargestellt und der Maximalpreis für das Abonnement kann abgeschätzt werden. Bei der Schätzung ist es dennoch wichtig, nicht nur die Ergebnisse der Umfrage zu analysieren, sondern auch die Preise der Konkurrenten in der gleichen Branche, wie MAXQDA[30], in Betracht zu ziehen.

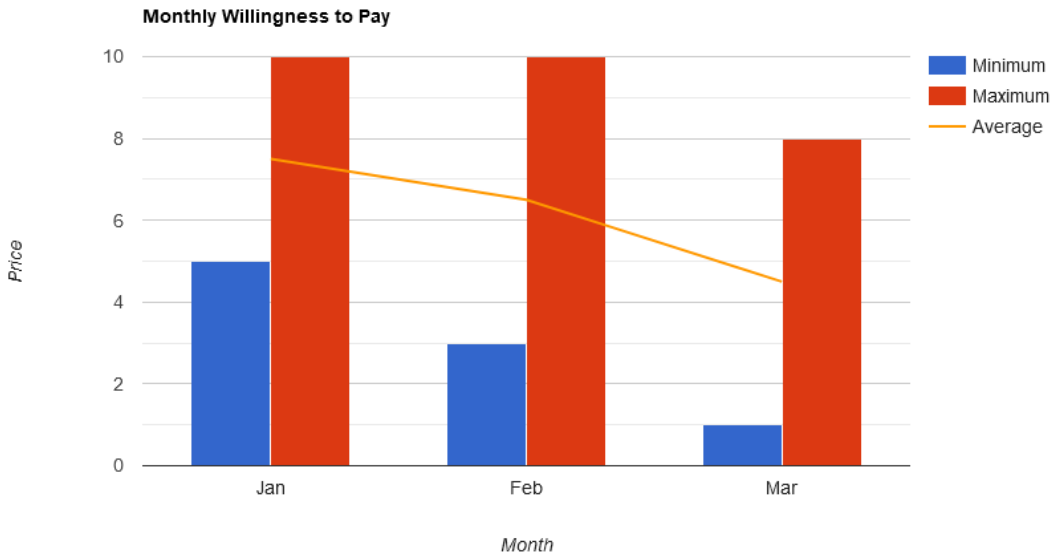

Abbildung 8.1 Beispieldiagramm für eine Willingness-To-Pay-Komponente

[28] https://qdacity.com/de/preise/
[29] https://developers.google.com/chart/interactive/docs/gallery/combochart?hl=de
[30] https://www.maxqda.com/de

Net Promoter Score

Der NPS, der 2003 von Fred Reichheld[31] erfunden wurde, gilt mit dem Customer Satisfaction Score und Customer Effort Score als einer der meistbenutzten Userfeedback Metriken, die von Unternehmen in der ganzen Welt gesammelt wird und wird als oft als Indikator für zukünftige Verkaufszahlen gesehen (Fitzgerald, 2017). Die Berechnung und das Einsammeln dieser Metrik wurde in Abschnitt 3.2 vorgestellt. Als Ort für die Komponente wäre das Personal Dashboard sinnvoll, da diese die zentrale Seite von QDAcity ist, über die immer navigiert werden muss. Da das Ergebnis der Komponente eine Zahl zwischen -100 und 100 ist, würde eine weitere *Valuebox* in den allgemeinen Statistiken reichen, um die Ergebnisse darzustellen. In Abbildung 8.2 wird ein Beispiel der resultierenden allgemeinen Statistiken gezeigt. Die Farbe und Icons sind frei wählbar.

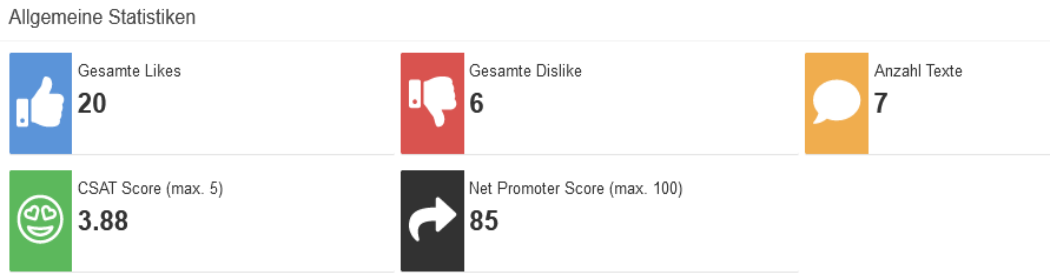

Abbildung 8.2 Beispiel für Net Promoter Score auf der Admin-Page

[31] https://www.netpromotersystem.com/about/

9 Zusammenfassung

Das Ziel dieser Arbeit war es, QDAcity um ein Userfeedback-System zu erweitern, um den Kunden einen einfachen und intuitiven Weg zu bieten, Feedback abzugeben und gleichzeitig dem Produktteam eine graphische Darstellung dieser aggregierten Daten auf der Admin-Page zu ermöglichen.

Im ersten Schritt wurde die Webanwendung QDAcity vorgestellt und die verschiedenen Seiten (Public-Page, Personal-Dashboard, Coding-Editor und Admin-Page) mit ihren Funktionen beschrieben. Zusätzlich wurden die Technologien vom Backend und Frontend von QDAcity vorgestellt.

Anschließend wurden die verschiedenen Arten von Benutzerfeedback vorgestellt und anhand ihrer aktuellen Relevanz eine Auswahl getroffen, welche Arten als Einstieg in ein Userfeedback-System geeignet sind.

Daraufhin wurden die funktionalen und nicht-funktionalen Anforderungen an das Userfeedback-System definiert. Es wurden insgesamt 13 funktionale und 7 nicht-funktionale Anforderungen definiert. Diese Anforderungen wurden mithilfe der konzeptionierten Architektur aus Kapitel 5 erfüllt, die beschreiben soll, wie die Komponenten im Frontend und Backend funktionieren.

Anschließend an die Konzeption wurde die Implementierung der Architektur in Kapitel 6 erläutert. Im ersten Schritt wurde die Umsetzung des Schichtenmodells im Backend erklärt und anschließend wurde auf die Implementierung und Design der Komponenten eingegangen.

Die Anforderungen aus Kapitel 4 wurden dann in Kapitel 7 nach Erfüllung evaluiert. Dabei konnten 20 von 22 Anforderungen erfüllt werden. Bei den übrigen zwei konnte eine aus zeitlichen Gründen nur teilweise erfüllt werden und die andere wurde aus analytischen Gründen anders umgesetzt.

Zum Schluss der Arbeit wurden in Kapitel 8 mögliche Erweiterungen des Userfeedback-Systems vorgestellt. Dabei wurde sowohl der Ort der Komponente als auch die graphische Darstellung auf der Admin-Page erklärt.

Zusammenfassend verfügt QDAcity nun über ein leicht erweiterbares Userfeedback-System. Mit den bereits implementierten Feedback-Komponenten können Benutzer nun leicht und intuitiv Feedback zu verschiedenen Seiten und Funktionalitäten der Webanwendung QDAcity geben. Diese Feedbacks können dann vom Produktteam auf der Admin-Page eingesehen und analysiert werden.

10 Literaturverzeichnis

Lehman, M. M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, *68*(9), 1060–1076. https://doi.org/10.1109/PROC.1980.11805

Reichheld, F. F. (2003, Dezember 1). The One Number You Need to Grow. *Harvard Business Review*. https://hbr.org/2003/12/the-one-number-you-need-to-grow

Reality Based Group. (2023, April 30). Using Customer Feedback to Drive Continuous Im provement. https://www.realitybasedgroup.com/all/using-customer-feedback-to-drive-continuous-improvement-in-your-business-operations-and-services/

Die SOPHISTen, H. (2016). *Schablonen für alle Fälle*.

Richards, M. (2015). *Software Architecture Patterns: Understanding Common Architecture* Patterns and when to Use Them. O'Reilly Media.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, *22 140*, 55–55.

Fitzgerald, M. (2017). Net Promoter - Implement the System: Advice and Experience from Leading Practitioners. (n.p.): Oswald Schier

Diller, H. (2008). *Preispolitik*. W. Kohlhammer Verlag.

Schuster, M. (2022, Oktober 14). Kontinuierliche Verbesserung auf die agile Art. *Echometer*. https://echometerapp.com/de/kontinuierliche-verbesserung-agile-art/

Team, S. C. (2022, November 25). Qualitative Vs Quantitative Feedback – Know The Difference. *Speak Ai*. https://speakai.co/qualitative-vs-quantitative-feedback/

Dixon, M., Freeman, K., & Toman, N. (2010, Juli 1). Stop Trying to Delight Your Customers. *Harvard Business Review*. https://hbr.org/2010/07/stop-trying-to-delight-your-customers

Haije, E. G. (2020, Oktober 14). *Der vollständige Guide zu User Feedback*. Mopinion. https://mopinion.com/de/user-feedback/

Haije, E. G. (2017, Februar 7). *Quantitative vs Qualitative Online Customer Feedback*. Mopinion. https://mopinion.com/quantitative-vs-qualitative-online-customer-feedback/

Hoffman, D., & Fodor, M. (2010). Can You Measure the ROI of Your Social Media Marketing? *MIT Sloan Management Review*, *52*.

Kotler, P., & Keller, K. L. (2006). *Marketing Management*. Pearson Prentice Hall.

Appendix

Methodenname	HTTP-Methodentyp
getTotalLikes	GET
getTotalDislikes	GET
getTotalTextFeedbacks	GET
getStatisfiedScore	GET
listUserFeedbacks	GET
deleteFeedback	DELETE
insertVotingFeedback	POST
updateVotingFeedback	PUT
insertTextFeedback	POST
insertSatisfiedFeedback	POST
getFeedbackFromTargets	GET
getTargetPageList	GET
getTargetSubjectList	GET
getTotalLikesForSubject	GET
getTotalDislikeForSubject	GET
getTextFeedbacks	GET

Tabelle 10.1 Alle REST-Endpunkte