Web App Analytics Warehouse with BigQuery

BACHELOR THESIS

Andreas May

Eingereicht am 15. April 2023

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. April 2023

Lizenz

Erlangen, 15. April 2023

Abstract

In established “Software as a Service” markets, user acquisition is one of the most important but at the same time most challenging tasks for new platforms. Both advertising campaigns and search engine optimization require a broad and reliable dataset as a basis for decision-making and optimizations, as well as an intuitive yet powerful interface for visualization and evaluation of this data. QDAcity is a competitor in the qualitative data analysis space, dealing with exactly this challenge. As part of this work, an analytics warehouse was planned and built specifically for QDAcity, tailored to the requirements and needs of the application. In addition, an interactive admin dashboard was designed and implemented using React, allowing for real time aggregation, evaluation and visualization of the collected data. In this context, the advantages and motivation for an in-house implementation compared to the purchase of existing external software are discussed as well.

Zusammenfassung

Nutzergewinnung ist für “Software As A Service” Anbieter in bereits etablierten Märkten eine der wichtigsten und gleichzeitig herausforderndsten Aufgaben. Sowohl für gezielte Werbekampagnen als auch für Suchmaschinenoptimierung wird eine breite und zuverlässige Datenmenge als Grundlage für das Treffen von Entscheidungen und Optimierungen benötigt, sowie eine intuitive aber gleichzeitig leistungsstarke Oberfläche für Visualisierung und Auswertung eben jener Daten. Einer dieser Anbieter ist QDAcity, eine Anwendung für qualitative Datenanalyse. Im Rahmen dieser Arbeit wurde für QDAcity ein Analytics Warehouse geplant und implementiert, welches auf die Anforderungen und Bedürfnisse der Plattform zugeschnitten ist. Ergänzend dazu wurde zudem ein interaktives Admin-Dashboard gestaltet und mittels React implementiert, welches die Aggregation, Auswertung und Visualisierung der gesammelten Daten in Echtzeit ermöglicht. In diesem Zusammenhang werden auch die Vorteile und Motivation für eine eigene Implementierung gegenüber einem Einkauf bereits existierender externer Software diskutiert.

Inhaltsverzeichnis

1 Einführung **1**
1.1 Wie funktionieren SEO und SEA? 1
1.2 Die Rolle von Analytics 2

2 Anforderungen **5**
2.1 Funktionale Anforderungen 5
2.1.1 Ablösung des bisherigen Systems 5
2.1.2 Wechsel der Datenbank hin zu BigQuery 5
2.1.3 Verbesserung der Liniendiagramme 5
2.1.4 Erkennung Lead-generierender Inhalte und Marketing Strategien 6
2.1.5 UX Verbesserungen 6
2.2 Nicht-Funktionale Anforderungen 7
2.2.1 Bedienbarkeit und Barrierefreiheit 7
2.2.2 Wartbarkeit und Erweiterbarkeit 7
2.2.3 Performance 7
2.2.4 Testbarkeit 8
2.2.5 Sicherheit 8

3 Architektur **9**
3.1 Aufbau des Analytics Systems 9
3.2 Auswahl der Datenbank 10
3.3 Planung der Datensammlungsschnittstelle 11
3.4 Planung der Datenauswertungsschnittstelle 12

4 Design und Implementierung **13**
4.1 Ablösung der bisherigen Infrastruktur 13
4.1.1 Gestaltung des Datenbankschemas 13
4.1.2 Erstellung der neuen Datensammlungsschnittstelle 16
4.1.3 Migration der Altdaten nach BigQuery 18
4.2 Visualisierung von Nutzerereignissen 18
4.2.1 Migration bisheriger Visualisierungen 18

4.2.2 Filter 21
4.2.3 Mehrdimensionale Visualisierungen 24
4.2.4 Weitere Anzeigeoptionen 25
4.2.5 Neue Visualisierungen 26
4.3 Visualisierung einzelner Nutzer und Navigationspfade 28
4.3.1 Übersicht kürzlich aktiver Nutzer 28
4.3.2 Nutzerdetailansicht 29

5 Auswertung **33**

6 Ausblick **35**

Literaturverzeichnis **37**

Abbildungsverzeichnis

3.1 Der Web-Analytics Prozess 9

4.1 Beispieldiagramm wie es im Dashboard zu sehen ist 20
4.2 Die Filterleiste 22
4.3 Anpassung der Attributfilter 23
4.4 Anpassung des Vergleichswertes 24
4.5 Graph der nach “userDevice” aufgeteilt wurde 25
4.6 Übersicht über die Anzeigeoptionen 26
4.7 Globale Seitenwechsel als Sankey Diagramm 27
4.8 Liste der kürzlichen Registrierungen 28
4.9 Kerninformationen über einen Benutzer 29
4.10 Sitzungsübersicht über einen Benutzer 30
4.11 Beispielhafter Weg eines Nutzers 30
4.12 Nutzerpfade ohne Ausrichtung 31
4.13 Weiterer Nutzerpfad mit vier Sitzungen 31

Akronyme

API Application Programming Interface, dt. Programmierschnittstelle

DSGVO Datenschutzgrundverordnung

GCP Google Cloud Platform, Cloud Angebot der Firma Google

KPI Key Performance Indicators, siehe Google (2024c)

REST REpresentational State Transfer, siehe Red Hat (2020)

SEA Search Engine Advertising, dt. Suchmaschinenwerbung

SEO Search Engine Optimization, dt. Suchmaschinenoptimierung

URL Uniform Resource Locator

UX User Experience, dt. Nutzererfahrung/Nutzererlebnis

WCAG W3C Accessibility Guidelines, siehe Spellman et al. (2023)

1 Einführung

QDAcity ist eine Anwendung für qualitative Datenanalyse, also der Auswertung von meist in Textform vorliegendem Datenmaterial im Hinblick auf eine bestimmte Fragestellung beziehungsweise einem konkreten Forschungsproblem (Springer, n. d.). Entwickelt wird diese an der Friedrich-Alexander Universität Erlangen-Nürnberg und findet gerade hier auch einen Großteil ihrer aktuellen Nutzerschaft. Viele Studenten nutzen QDAcity für ihre Forschungsarbeiten entweder privat oder im Rahmen diverser Kurse und Projekte.

Weniger bekannt war QDAcity bisher hingegen lange Zeit außerhalb der Universität. Zwar hat die Plattform bereits einige externe Nutzer ansprechen können, jedoch stellt sich die Nutzergewinnung im Allgemeinen, wie bei vielen Start-ups von ähnlicher Größe, als Herausforderung dar. Zusätzlich gibt es im Bereich der qualitativen Datenanalyse bereits einige etablierte Plattformen.

Um neue Nutzer zu gewinnen, muss QDAcity also nicht nur neue Interessenten, die sich bisher noch auf keiner der Plattformen angemeldet haben, von der eigenen überzeugen, sondern im Idealfall auch Bestandskunden von Konkurrenten abwerben. Dafür setzt QDAcity neben Präsenz auf sozialen Medien vor allem auf die etablierten Methoden der Suchmaschinenoptimierung (SEO) und der gezielten Suchmaschinenwerbung (SEA).

1.1 Wie funktionieren SEO und SEA?

Suchmaschinenoptimierung und Suchmaschinenwerbung sind sich im Kern sehr ähnlich. Ziel ist es bei beiden, bei bestimmten Suchanfragen hohe Positionen auf der jeweiligen Suchergebnisseite zu erreichen und dies wiederum mit Inhalten oder Seiten, die jeweils möglichst gut zur Suchanfrage des Nutzers passen. Unterschiedlich ist jedoch die Herangehensweise.

Um mit SEO hohe Positionen zu erzielen, muss die eigene Seite in jeglicher Hinsicht optimiert sein. Sie sollte schnell laden und einfach zu bedienen sein. Die Inhalte sollten logisch und sinnvoll strukturiert, sowie visuell ansprechend sein. Die richtige Länge von Überschriften und Seitentiteln, sowie Metadaten müssen

ebenfalls beachtet werden. So beschreibt es Google (2024c). Allerdings schaffen es jedoch die meisten QDAcity-Konkurrenten hier zu Punkten, da diese ebenfalls auf eben genannte Kriterien achten.

Die Entscheidung, wer den begehrten ersten Platz in den Suchergebnissen erhält, hängt also immer mehr davon ab, wie gut der Inhalt zur jeweiligen Suchanfrage passt. Möchte man bessere Chancen haben und mehr Suchbegriffe abdecken, muss man als Seitenbetreiber entweder mehr Inhalt generieren und bereitstellen oder vorhanden Inhalt spezifischer gestalten und optimieren. Ein Großteil von modernem SEO ist folglich das Betreiben eines Blogs oder etwas Vergleichbarem, auf welchem man eine Vielzahl von Artikeln veröffentlicht, welche alle jeweils ganz bestimmte Fragestellungen oder Thematiken abdecken, sodass auch für sehr spezifische Suchanfragen ein passender Artikel vorhanden ist. Wichtig für den Webseitenbetreiber ist hier nicht nur zu wissen, welche relevanten Suchbegriffe beziehungsweise Schlüsselwörter häufig gesucht werden, um dafür Artikel zu erstellen, sondern auch, wie gut die eigenen Artikel auf den jeweiligen Suchergebnisseiten abschneiden.

Wo bei SEO eher breitere Zielgruppen abgedeckt und viele Begriffe in den Fokus genommen werden können, zielen die meisten SEA Kampagnen meist auf spezifischere Themen oder Personengruppen ab. Hier sind ebenfalls Suchbegriffe und Schlüsselwörter sehr wichtig. Werbekampagnen werden auf bestimmte Begriffe ausgelegt. Nur wenn diese mit der Suchanfrage übereinstimmen, wird auch die eigene Werbung gezeigt. Bei diesen Begriffen möchte man nun weder zu spezifisch werden, weil die gewählten Formulierungen so vielleicht fast nie gesucht werden, noch möchte man zu viele Begriffe abdecken, weil man schnell Gefahr läuft auf irrelevanten Suchanfragen zu landen oder den falschen Inhalt zu bewerben. Beide Situationen kosten natürlich trotzdem Geld, obwohl die Wahrscheinlichkeit Nutzer zu gewinnen jeweils sehr gering ist.

1.2 Die Rolle von Analytics

Bei beiden Methoden wird also großer Wert auf die richtigen Schlüsselwörter und Suchbegriffe gesetzt. Man möchte als Webseitenbetreiber jedoch nicht nur herausfinden, welche Begriffe häufig gesucht werden und zu Seitenaufrufen führen, sondern vielmehr, wie gut diese jeweils im Durchschnitt zu einer Konvertierung führen.

Eine Konvertierung wird es genannt, wenn ein Nutzer eine eigens definierte gewünschte Aktion ausführt (Bekavac & Praničević, 2015). Das könnte etwa eine abgeschlossene Zahlung sein, ein Anmelden für einen Newsletter, oder im Fall von QDAcity das Registrieren eines Nutzeraccounts.

Das Gegenteil einer Konvertierung wird Bounce genannt, also das Verlassen der Seite, ohne dass die gewünschte Aktion ausgeführt wurde (Bekavac & Praničević, 2015).

Hier kommt nun Analytics ins Spiel. Man möchte ein System erschaffen, welches einen Überblick über die eigenen Besucher schafft. Wie viele konvertieren, wie viele bouncen und welchen Zusammenhang hat das jeweilige Verhalten mit anderen Interaktionen auf der Seite? Gibt es beispielsweise kaputte Links oder Buttons, die sicher zu einem Bounce führen und deswegen dringend repariert werden müssen? Oder existieren Artikel, die zwar hohe Positionen in Suchergebnissen erzielen, aber keine einzige Konvertierung erzielen? Um Fragen wie diese beantworten zu können, bedarf es einer soliden Datengrundlage in Form eines Analytics "Data Warehouse", das von dem System konstant befüllt und bei Bedarf ausgewertet werden kann.

Nun sind die oben genannten Probleme und Fragestellungen natürlich nicht neu. Fast jede "Software As A Service" Platform (Salesforce, n. d.), wie auch QDAcity, hat diese Anliegen und so gibt es bereits hunderte Produkte, die genau hier helfen wollen. Im Rahmen dieser Arbeit wird jedoch ein derartiges Analytics-System von Grund auf neu entwickelt, was die Frage stellt, wieso man sich nicht einfach für eines dieser bereits bestehenden Produkte entschieden hat.

Eine maßgeschneiderte Lösung ermöglicht zunächst eine genauere Kontrolle über den Umfang und die Speicherung der erhobenen Daten. Dies gewährleistet nicht nur die Einhaltung der DSGVO, was viele externe Anbieter nicht können oder wollen, sondern vermeidet auch die Bindung an bestimmte Anbieter. Das wiederum garantiert niedrige Laufzeitkosten des Systems, da man nicht von Preiserhöhungen der Drittanbieter abhängig ist und auch keine Servicegebühren oder sonstige Nebenkosten an eben jene abgeben muss.

Ebenso muss beachtet werden, dass die meisten, für kleine und mittelgroße Unternehmen verfügbaren Analyseplattformen, Schwierigkeiten haben, Werbe- und Tracking-Blockern entgegenzuwirken. Hier hat das Betreiben einer nicht-standardisierten beziehungsweise nicht öffentlich verfügbaren Lösung den klaren Vorteil, dass man das eigene System einfach so gestalten kann, dass es von solchen Browsererweiterungen nicht entdeckt und folglich nicht blockiert wird.

Abschließend erlaubt die maßgeschneiderte Implementierung eine bessere Integration in die bereits bestehenden Systeme der Plattform. Neben Integration von Auswertung und Darstellung der Ergebnisse in bereits bestehende Dashboards und Tools können auch andere Datenquellen, welche nicht direkt durch das Analytics-System erhoben werden, so mit in die Auswertung einfließen.

2 Anforderungen

Vor der Implementierung wurden einige funktionale sowie nicht-funktionale Anforderungen festgehalten.

2.1 Funktionale Anforderungen

2.1.1 Ablösung des bisherigen Systems

Schon vor Beginn der Arbeit existierte ein rudimentäres Analytics-System mit einem einfachen Dashboard. Dieses Dashboard stellte einige wichtige numerische Werte übersichtlich dar und enthielt eine kleine Anzahl an Diagrammen, die den Verlauf bestimmter Metriken über die Zeit visualisierten. Grundlage für alle folgenden Anforderungen ist es, dass die neue Implementierung diese bisherige Funktionalität im vollen Umfang ablöst und funktional komplett ersetzt.

2.1.2 Wechsel der Datenbank hin zu BigQuery

Ebenso grundlegend ist der Wechsel der Datenbank hin zu BigQuery, ein von Google entwickeltes Data Warehouse (Google, 2024a). Das bisherige System basierte auf "Datastore", eine ebenfalls von Google entwickelte NoSQL-Datenbank (Google, 2024b), die jedoch im Hinblick auf unseren Anwendungsfall schnell an ihre Grenzen stößt. In diesem Zusammenhang ist eine Migration der vorhandenen Daten in den neuen BigQuery-Datenpool erforderlich, da die mit dem bisherigen System gesammelten Daten nicht verloren gehen sollen.

2.1.3 Verbesserung der Liniendiagramme

Das bisherige Dashboard enthielt eine kleine Anzahl von Diagrammen zur Visualisierung zeitlicher Verläufe verschiedener Metriken. Dabei handelte es sich um einfache Liniendiagramme, die die Zeit auf der X-Achse und den entsprechenden Wert auf der Y-Achse darstellten.

Verlangt ist nun, dass jeder Graph die Möglichkeit bietet, mehrere Datensätze gleichzeitig anzuzeigen, um eine Metrik anhand verschiedener Attribute vergleichen zu können. Bei einem Graphen, der die Anzahl der Nutzer pro Tag zeigt, soll es beispielsweise möglich sein, diesen in zwei separate Linien aufzuteilen, welche jeweils ausschließlich Smartphone-Nutzer beziehungsweise ausschließlich Laptop-Nutzer zählen.

Hierbei soll es dem Betrachter möglich sein, mehrere Datensätze entweder voneinander unabhängig oder gestapelt darzustellen, sowie die Option zu haben, die Darstellung von Liniendiagramm zu einem Balkendiagramm zu ändern.

2.1.4 Erkennung Lead-generierender Inhalte und Marketing Strategien

Das Ziel des Analytics-Warehouses ist die Optimierung der Nutzergewinnungsstrategien. Darauf soll bei der Überarbeitung des Dashboards großer Wert gelegt werden.

So sollen neue Visualisierungen hinzugefügt werden, welche beim Verstehen von Nutzer-Pfaden, also den Wegen, die die Nutzer durch die Webseite nehmen, helfen. Ebenso soll es möglich sein, hier einzelne Nutzer gezielt anzuzeigen, um deren Interaktionen mit der Website unabhängig nachvollziehen zu können.

Dafür ist es notwendig individuelle Nutzer über einen gewissen Zeitraum wiederzuerkennen, selbst wenn sich diese noch nicht für ein Nutzerkonto angemeldet haben. Neben einem solchen identifizierenden Erkennungsmerkmal sollen noch weitere Informationen über den Besuch gesammelt werden. Dazu zählt der Eintrittspunkt, also welche Seite oder welchen Artikel für das Betreten der Webseite verantwortlich ist, sowie Metadaten zum Eintritt, wie die Kampagne, den "Referer", und die "Value Track parameter", auf welche später noch im Detail eingegangen wird. Nach dem Eintritt soll jede Interaktion des Nutzers mit der Webseite festgehalten werden, wie etwa ein Wechsel auf eine andere Seite, oder ein klick auf einen Knopf.

2.1.5 UX Verbesserungen

Eine Verbesserung der "User Experience" (UX), also des Nutzererlebnisses, beschreibt eine Verbesserung der Bedienbarkeit der Seite. Hier im Hinblick auf das Dashboard, mit dem Ziel die Produktivität der Dashboard-Nutzer (im Folgenden auch als Analytiker bezeichnet) zu steigern. So sollen Zeitfenster und Filter, welche bisher pro Graph in mehreren Klicks angewendet werden mussten, vereinfacht und vereinheitlicht werden. Ebenso sollen individuelle Profile für jeden Analytiker eingerichtet werden, sodass sich jeder das eigene Dashboard im Hinblick auf Tastenkombinationen oder Standardeinstellungen individuell einrichten

kann. Abschließend soll es möglich sein, in einem visuellen Editor und ohne Code schreiben zu müssen, einzelne Graphen in ihrer Funktionsweise zu verändern oder gar komplett neue Graphen direkt im Dashboard hinzuzufügen.

2.2 Nicht-Funktionale Anforderungen

2.2.1 Bedienbarkeit und Barrierefreiheit

Alle neuen Oberflächen sollen das bereits existierte Farbschema sowie die Designrichtlinien von QDAcity verwenden. Besonderer Fokus liegt hier auch darauf, die "Web Content Accessibility Guidelines" (WCAG) in der Version 3.0 zu erfüllen. WCAG definiert wichtige Richtlinien bezüglich Kontrast, Text Formatierung, Layout und vielem mehr, um Webseiten für Menschen mit verschiedensten Behinderungen zugänglich zu machen (Spellman et al., 2023). Zusätzlich sollen alle neu hinzugefügten Texte in alle von QDAcity unterstützten Sprachen übersetzt und lokalisiert werden.

2.2.2 Wartbarkeit und Erweiterbarkeit

Das neue Dashboard soll leicht zu erweitern sein. Sollten in Zukunft neue Arten von Diagrammen oder Visualisierungen hinzugefügt werden müssen, muss dies ohne großen Mehraufwand möglich sein. Diesbezüglich soll das neue Dashboard, sofern möglich, bereits in anderen Teilen der Platform implementierte Komponenten wiederverwenden. Neue Funktionalität wiederum sollte so in Komponenten auslagert werden, dass diese auch in anderen Teilen der Anwendung wiederverwendet werden kann. React Komponenten sollen gemäß aktueller Standards komplett funktional implementiert werden.

Das Prinzip der "separation of concerns", also einer Trennung des Codes in Endpunkt, Controller und Datenbank, sollte jederzeit eingehalten werden.

Neue Schnittstellen sollen einheitlich gestaltet, klar definiert und dokumentiert werden.

2.2.3 Performance

Neue Seiten, darunter das Dashboard, sollten in unter 0,5 Sekunden laden. Graphen und andere Visualisierungen sollten in maximal 2,5 Sekunden nach Anfrage sichtbar sein (Walton & Pollard, 2024).

Datenbankanfragen mit denselben Optionen sollten nicht mehrfach hintereinander ausgeführt werden. Solche Anfragen sollen stattdessen für einen Zeitraum zwischengespeichert werden, um die Datenbank zu entlasten und Dashboard-Anfragen zu beschleunigen.

2.2.4 Testbarkeit

Zu besseren Wartbarkeit und Erweiterbarkeit zählt auch das automatisierte Testen der einzelnen Komponenten. So soll neuer Code eine Testabdeckung durch Unit-Tests von mindestens 90% haben. Alle neuen Funktionalitäten sollten zusätzlich in einem Acceptance-Test berücksichtigt sein.

2.2.5 Sicherheit

Auch wenn alle gesammelten Daten anonym und nicht auf reale Personen zurückzuführen sind, dürfen sie nur für den angegebenen Zweck verwendet werden. Das bedeutet auch, alle gesammelten Daten müssen ausreichend vor unbefugtem Zugriff geschützt werden. Sie sollen ausschließlich über das Dashboard zugänglich sein, welches folglich ebenso geschützt und ausschließlich für Analytiker zugänglich gemacht werden darf.

3 Architektur

3.1 Aufbau des Analytics Systems

Ein Analytics System kann grundsätzlich viele verschiedene Formen annehmen. Waisberg und Kaushik (2009) schlagen deshalb folgendes grundlegendes Modell zur Gestaltung eines solchen Systems vor:

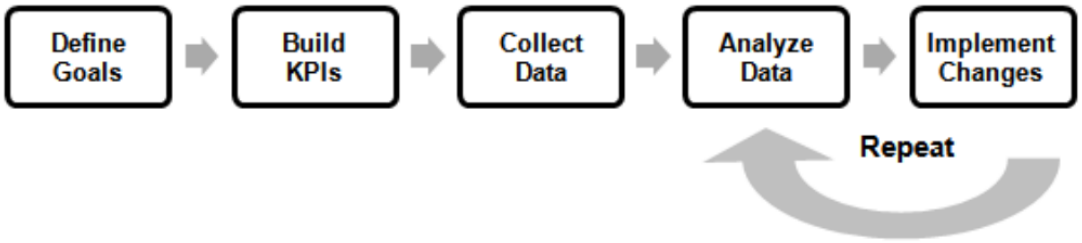


Abbildung 3.1: Der Web-Analytics Prozess

Als ersten Schritt nennen Waisberg und Kaushik (2009) eine Festlegung auf Ziele, die mit dem System erreicht werden sollen. Im Fall von QDAcity ist dies, wie eingangs beschrieben, die Nutzergewinnung.

Daraufhin müssen "Key Performance Indicators", kurz KPIs, definiert werden. Einen solchen KPI könnte beispielsweise das Besuchen der Registrierungsseite oder das Registrieren selbst darstellen. Dies sind beide konkrete Ereignisse, die das übergeordnete und absichtlich unpräzise definierte Ziel messbar machen.

Der dritte Schritt trägt den Titel "Daten sammeln". Hier schlagen Waisberg und Kaushik (2009) zwei für QDAcity infrage kommende Ansätze vor.

"JavaScript Tagging", das Einbetten von extra für Analytics zuständigem JavaScript Code auf der Webseite, und "Web Logs", ein rein serverseitiges Aufzeichnen aller Netzwerkanfragen.

Da jedoch auch Interaktionen mit der Website aufgezeichnet werden sollen, die nicht explizit mit einem Seitenwechsel verbunden sind oder durch sonstige Nebeneffekte eine Netzwerkanfrage an den Server schicken, fiel die Wahl auf "JavaScript Tagging".

Art und Aufbau der gesammelten Daten

Nguyen et al. (2017) beschreiben einen Ansatz namens "Atomic User Interactions", zu Deutsch "Atomare Nutzerinteraktionen". Der Gedanke ist hier, jegliche Interaktionen des Nutzers atomar, also voneinander unabhängig und so grundlegend wie möglich, einzeln aufzuzeichnen. Neben Aufrufen der Seite und Drücken eines Knopfes kann so gut wie alles als Interaktion definiert werden, solange dieses von anderen Ereignissen und Interaktionen unabhängig ist. So wird im Idealfall der gesamte Aufenthalt des Nutzers auf der eigenen Seite atomar festgehalten und kann theoretisch komplett nachvollzogen werden.

Auch wenn eine Auswertung der auf diese Art gesammelten Daten gegebenenfalls komplexer ausfällt, als eine Auswertung von bereits aggregierten Daten, oder Daten, die nur eine kleine Anzahl expliziter Ereignisse widerspiegeln, hat das System der atomaren Nutzerinteraktionen einen entscheidenden Vorteil. Da jegliche Interaktion aufgezeichnet ist, können so gut wie alle Fragestellungen im Nachhinein analysiert und ausgewertet werden, ohne dass dafür eine Änderung an der Implementierung des Systems notwendig ist. So können auch neue Fragestellungen oder Metriken problemlos an historischen Daten untersucht werden.

Konkreter: Sollte QDAcity in Zukunft das eigene Ziel von Nutzergewinnung abwenden und andere KPIs verfolgen, werden bereits gesammelte Daten nicht nutzlos. Neue Auswertungen und Analysen können direkt auf dem vorhandenen System aufbauen.

3.2 Auswahl der Datenbank

Auch wenn die Wahl der Datenbank, wie schon in den Anforderungen festgehalten, ultimativ auf BigQuery fiel, stand diese Entscheidung nicht von Anfang an fest. Es gab einige Anforderungen, welche die Datenbank erfüllen musste, um in Erwägung gezogen zu werden.

Die Datenbank muss zunächst mit sehr großen Datenmengen zurechtkommen können und trotzdem in Echtzeit abfragbar bleiben. Pro Sitzung generiert ein Nutzer häufig dutzende Ereignisse, jedes davon ein eigener Datenbankeintrag. Die Abfragezeit darf durch solche Datenmengen nicht beeinträchtigt werden.

Nachdem alle Ereignistypen in derselben Datenbank gespeichert werden und so zwangsläufig dasselbe Schema einhalten, jedoch nicht immer dieselben Daten mit

sich führen, führt das zu einem sogenannten "Sparse Dataset", also einem Datensatz mit vielen "null" Werten (Simic, 2024). Die Datenbank muss damit ebenfalls gut zurechtkommen, im besten Fall sogar darauf optimiert sein.

Das alles wiederum muss in einem kostengünstigen Paket verfügbar sein.

BigQuery erfüllt alle eben genannten Anforderungen, da es speziell für Anwendungsfälle wie unseren entwickelt wurde. BigQuery ist außerdem eine zukunftssichere Entscheidung, da es von einem der größten Cloud-Provider, Google, entwickelt wird und neben namhaften Kunden wie Spotify (Google Cloud, 2020) oder The New York Times (Podojil, 2021) auch anzunehmen ist, dass Google selbst interne Prozesse auf dieser Technologie aufbaut. Als zusätzliche Absicherung erlaubt BigQuery außerdem den Massenimport und -export von Daten, sodass Bedenken zu sogenanntem "Vendor lock-in" (Mandarina & Hase, 2019) vernachlässigbar sind.

Darüber hinaus ist BigQuery ein Angebot der Google Cloud Platform (GCP), welche bereits das restliche QDAcity Projekt beherbergt. Das wiederum vereinfacht nicht nur den Datentransfer massiv, da die Daten nie die Google Cloud verlassen müssen, sondern integriert auch die Bezahlung, Kosten- und Nutzerverwaltung und Monitoring einfach in das restliche Projekt. Abschließend erlaubt BigQuery zusätzliche Datenimporte aus anderen Quellen innerhalb der GCP, wie beispielsweise der "Google Search Console", um zukünftig noch weitere Datenquellen in das Warehouse einfließen zu lassen.

3.3 Planung der Datensammlungsschnittstelle

Um die einzelnen Ereignisse von dem Frontend, wo sie entstehen, in das Backend, wo sie anschließend aufgezeichnet werden, zu befördern, wird eine Schnittstelle benötigt. Diese sollte zur besseren Wartbarkeit den REST (Red Hat, 2020) Standard erfüllen, da das restliche QDAcity-Projekt bereits auf diesem aufbaut. Auch wenn die Schnittstelle in ihrem Umfang sehr überschaubar ist, müssen zwei Anliegen im Zusammenhang mit gängigen Browsererweiterungen beachtet werden.

So muss für die Schnittstelle eine URL gewählt werden, welche nicht sofort als Endpunkt für Tracking erkennbar ist. Begriffe wie etwa "analytics" oder "tracking" sollten vermieden werden, da diese mit hoher Wahrscheinlichkeit standardmäßig von einigen Adblockern blockiert werden. Ebenso der Payload, also die Daten, welche über diese Schnittstelle übertragen werden. Auch dieser muss sorgfältig gewählt werden, da auch hier bestimmte Schlüsselwörter, Inhalte oder Schemata automatisch erkannt und blockiert werden können.

3.4 Planung der Datenauswertungsschnittstelle

Als Gegenstück zur Datensammlungsschnittstelle muss eine weitere Schnittstelle zum Abfragen der gesammelten Daten erstellt werden, welche dann vom Dashboard für Auswertungen und Visualisierungen genutzt werden kann. Diese Schnittstelle ist vom Umfang allerdings etwas komplexer als ihr Gegenstück, da ein einfaches Zurückgeben aller Ereignisse auf Anfrage nicht ausreichend ist. Die großen Datenmengen von vielen tausenden Ereignissen würden neben langen Lade- und Wartezeiten auch das Dashboard beziehungsweise vielmehr den Browser des Analytikers überfordern. Es ist also essenziell, dass die Daten bereits im Backend ausgewertet und nur die aggregierten Werte an das Dashboard übergeben werden.

Die Entscheidung wurde getroffen, die Schnittstelle aus mehreren Endpunkten aufzubauen, jeweils einer pro Visualisierung im Dashboard. So kann je nach Anfrage eine andere Aggregation im Hintergrund ablaufen. Der Rückgabewert der unterschiedlichen Endpunkte ist jedoch immer vom Typ "Diagram", also einem generischen Diagramm, welches wiederum dann neben den Daten auch den eigenen Typen mit sich bringt. So muss auf Schnittstellen-Ebene nicht zwischen Zeitreihendiagrammen, Flussdiagrammen oder anderen unterscheiden werden. Das Dashboard wiederum kann alle Diagrammtypen unterscheiden und korrekt darstellen, ohne dass für jede Visualisierung eigener Code geschrieben werden muss, und ohne dass dem Dashboard für jede Visualisierung bekannt sein muss, um welche Art Diagramm es sich handelt. Um die Endpunkte noch generischer zu gestalten, wurde ebenfalls entschieden, alle häufig auftretenden Parameter bei Abfragen, wie etwa den betrachteten Zeitrahmen oder Filterfunktionen, zu standardisieren. So sind folglich alle Endpunkte in ihrer Bedienweise gleich, sie akzeptieren dieselben Eingaben und generieren Ausgaben in derselben Form, was Wartung und Erweiterbarkeit steigert.

Einzige Ausnahme bilden eine kleine Anzahl an Endpunkten für Metadaten, die Informationen zu aktuellen Kampagnen oder neu registrierten Benutzern anstelle eines Graphen zurückgeben. Diese Daten sollen im Dashboard für eine verbesserte Benutzererfahrung sorgen, indem bestimmte Felder, beispielsweise für das Filtern oder Sortieren von Datensätzen, Werte vorgeschlagen bekommen.

4 Design und Implementierung

4.1 Ablösung der bisherigen Infrastruktur

4.1.1 Gestaltung des Datenbankschemas

Auch wenn verschiedene Arten von Ereignissen protokolliert werden sollen, müssen diese in derselben Tabelle gespeichert werden und folglich demselben Schema folgen. Das wiederum bedeutet zwangsläufig, dass viele der Felder häufig Nullwerte annehmen werden, da sie nur von bestimmten Ereignistypen unterstützt werden. Somit können wir die Felder des neuen Schemas in zwei Gruppen unterteilen.

Erforderliche Felder (keine Nullwerte erlaubt)

Die Liste der erforderlichen Felder wurde so klein wie möglich gewählt, da auch Ereignisse, bei denen einige Daten fehlen, Informationsgehalt tragen und deswegen nicht verworfen werden dürfen. Ausschließlich Felder, welche absolut essenziell sind oder durch das Backend gesetzt werden und damit nicht fehlen können, sind im Schema als `required` markiert.

So benötigt jedes Ereignis eine eindeutige `id` als Primärschlüssel. Diese ID wird jedoch anderweitig nicht verwendet und sagt nichts über das Ereignis aus. Anders der Zeitstempel `datetime`, welcher den Ereigniszeitpunkt auf die Millisekunde genau festhält.

Erforderlich ist außerdem der `type`, also Typ des Ereignisses, sowie die `apiVersion`, also die Version der Schnittstelle, über welche das Ereignis übermittelt wurde.

Optionale Felder (Nullwerte erlaubt)

Die optionalen Felder lassen sich wiederum in zwei Gruppen unterteilen. Die, die den Benutzer beschreiben, welcher für das Ereignis verantwortlich ist, und nicht in Zusammenhang mit dem Ereignistypen stehen, sowie die, die Informationen

über das Ereignis selbst oder den Kontext des Ereignisses beschreiben und je nach Typen eventuell keine Verwendung finden.

Über den Nutzer wird dessen `userId` gespeichert, falls er eingeloggt ist, sowie eine `userVisitorId`, für welche er nicht eingeloggt sein muss. Letztere wird im Browser des Nutzers zufällig generiert und auch dort gespeichert. Es handelt sich hierbei um eine UUID (Leach et al., 2005) und sie erlaubt es später in der Auswertung wiederkehrende Nutzer zu erkennen. Eine zusätzliche `userSession` ID existiert analog zur Visitor ID. Auch sie wird direkt im Browser generiert, hat allerdings den entscheidenden Unterschied, dass sie ausschließlich im SessionStorage (Mozilla, 2023d) gespeichert und so nach Inaktivität des Nutzers automatisch gelöscht wird. Das erlaubt im Nachhinein einzelne Sitzungen desselben Nutzers auseinanderzuhalten.

`userCountry` trägt Informationen über den physischen Ort der jeweiligen Person. Für die Ermittlung dessen gibt es mehrere Ansätze. Eine Nutzung der "Geolocation API" (Mozilla, 2023b) wurde ausgeschlossen, da diese explizite Einwilligung des Nutzers benötigt und deren auf wenige Meter exakte Präzision für unseren Anwendungsfall nicht notwendig ist. Ebenso ausgeschlossen wurde eine Ermittlung über die IP-Adresse, da hier externe Dienste in Anspruch genommen werden müssten, die mit weiteren Kosten sowie Datenschutzbedenken verbunden wären. So wurde letzten Endes beschlossen, das Land über die Zeitzone zu ermitteln, welche mit JavaScript im Browser einfach abzurufen ist. Beim Abschicken eines Events setzt der Client den Wert von `userCountry` einfach auf dessen Zeitzone, welche dann serverseitig zu einem ISO 3166 Ländercode aufgelöst wird.

`userDevice` übermittelt die Größenordnung der Bildschirmauflösung. Angefangen bei `"mobile"` für Smartphones und sehr kleine Bildschirme, über `"tablet"` und `"desktop"` hin zu `"desktop_xl"` für hochaufgelöste große Monitore.

Das letzte Nutzer-Spezifische Feld trägt den Namen `userAgent` und protokolliert den gleichnamigen `"User-Agent"` HTTP-Header, welcher bei jeder Anfrage automatisch mitgeschickt wird und den Browser des Nutzers beschreibt (Mozilla, 2023c). Dieser Wert wird in unserem Fall dafür genutzt unnatürliche Seitenaufrufe, wie beispielsweise durch Bots oder Crawler (Cloudflare, n. d.), erkennen zu können.

Die Ereignis-beschreibenden Felder beginnen mit `dataElement`, welches die ID des Webseiten-Elements beschreibt, das das jeweilige Event ausgelöst hat. Hier kann es sich beispielsweise um einen Knopf oder Link handeln, der Wert kann aber auch "null" annehmen, falls das Ereignis nicht durch Interaktion mit einem bestimmten Element entstanden ist.

Die URL der Seite, auf der sich der Nutzer zum Zeitpunkt des Ereignisses befindet, wird unter `dataUrl` protokolliert.

Betritt ein Nutzer die Webseite über eine Werbekampagne, so wird in der URL der Parameter `?campaign=` vorhanden sein, der wiederum die jeweilige Kampagne identifiziert. Falls dies der Fall ist, wird die Kampagne aus der URL entfernt und separat als `dataCampaign` gespeichert.

`dataVTKeyword` und `dataVTType` sind verwandte Felder, die jeweils, falls vorhanden, die `?kw=` und `?matchtype=` URL-Parameter der Sitzung speichern. Hierbei handelt es sich um Metadaten von "ValueTrack", einer Funktion von Google Ads um mehr Informationen über die Suchanfrage des Nutzers zu erfahren, welche zum Klick auf die eigene Werbung geführt hat (Google, n. d. b).

`dataSessionOrigin` spiegelt den Wert des JavaScript-Objekts `Document.referrer` wider, welches wiederum die URL der Seite beinhaltet, die den Nutzer auf QDAcity verlinkt hat (Mozilla, 2023a).

Sollte ein Nutzer beim Arbeiten in einem Projekt ein Ereignis auslösen, so werden die ID und der Typ des Projektes als `dataProjectId` und `dataProjectType` gespeichert.

Ähnlich zu Projekten existieren Tutorials, eine Funktion von QDAcity, die Nutzern die App vorstellt und erklärt. Falls sich der Nutzer aktuell in einem Tutorial befindet, beinhalten die Werte von `dataTutorialId`, `dataTutorialStep` und `dataTutorialType` Informationen über dieses.

Resultierendes Schema

Zusammengefasst schaut das resultierende Schema wie folgt aus:

```
TABLE AnalyticsEvent (
  id INTEGER PRIMARY KEY,
  datetime DATETIME NOT NULL,
  type STRING NOT NULL,
  apiVersion STRING NOT NULL,
  userId STRING,
  userVisitorId STRING,
  userSession STRING,
  userCountry STRING,
  userDevice STRING,
  userAgent STRING,
  dataElement STRING,
  dataUrl STRING,
  dataCampaign STRING,
  dataVTKeyword STRING,
  dataVTMatchtype STRING,
  dataSessionOrigin STRING,
  dataTutorialId INTEGER,
  dataTutorialType STRING,
  dataTutorialStep INTEGER,
  dataProjectId INTEGER,
  dataProjectType STRING
)
```

Analog zu dem Schema wurde eine entsprechende Java Klasse namens `AnalyticsEvent` erstellt, die alle diese Werte kapselt und einfachen Datenbank Schreibzugriff gewährt. So reicht es eine neue Instanz der Klasse zu erstellen und die gewünschten Werte zu setzten. Mittels der `insert()` Methode können diese dann nach BigQuery gespeichert werden, ohne dass eine SQL-Anfrage formuliert werden muss.

4.1.2 Erstellung der neuen Datensammlungsschnittstelle

Für die Datensammlung wurde der vorhandenen QDAcity API ein neuer Endpunkt hinzugefügt, der per HTTP `POST` Request Daten entgegennimmt. Als URL-Pfad wurde nach einigen Tests `/system/record` festgelegt, da diese generisch genug ist, um von keinem der gängigen Adblocker blockiert zu werden. Im Request Payload befindet sich ein JSON-Objekt mit einem einzigen Feld namens `string`, welches wiederum das eigentliche Ereignis als Base64-kodierten Text enthält. Der extra Schritt der Base64-Kodierung ist eine weitere Maßnahme um vor aggressiv konfigurierten Adblockern versteckt zu bleiben und hat sich nach einigen Tests als sehr effektiv herausgestellt.

```
{
  "string": "eyJ0eXBlIjoiY2xpY2siLCJkYXRhRWxlbWVudCI6Iklu
             ZGV4LUhhc2hsaW5rLVRyYW5zY3JpcHRpb24iLCJ1c2Vy
             VmlzaXRvcklkIjoiMTYxMmI3MjUtODdlNi00Zjk4LTgw
             ZTYtMGJlMTI3MTU2YjkyIiwidXNlclNlc3Npb24iOiI5
             OTI0ZTYyMS01ODhkLTQ5OGUtOTllYi0xOGJkMjk0Mjc0
             ODAiLCJ1c2VyQ291bnRyeSI6IkV1cm9wZS9CZXJsaW4i
             LCJ1c2VyRGV2aWNlIjoiZGVza3RvcCIsImRhdGFVcmwi
             OiJodHRwczovL3FkYWNpdHkuY29tLyIsImRhdGFDYW1w
             YWlnbiI6bnVsbCwiZGF0YVZUS2V5d29yZCI6bnVsbCwi
             ZGF0YVZUTWF0Y2h0eXBlIjpudWxsLCJkYXRhU2Vzc2lv
             bk9yaWdpbiI6IiJ9"
}
```


Listing 4.1: Beispielhafter Payload eines POST /system/record Requests

Das Event ist wiederum selbst ein JSON-Objekt und folgt dem oben definierten Schema.


```
{
  "type": "click",
  "dataElement": "Index-Hashlink-Transcription",
  "userVisitorId": "1612b725-87e6-4f98-80e6-0be127156b92",
  "userSession": "9924e621-588d-498e-99eb-18bd29427480",
  "userCountry": "Europe/Berlin",
  "userDevice": "desktop",
  "dataUrl": "https://qdacity.com/",
  "dataCampaign": null,
  "dataVTKeyword": null,
  "dataVTMatchtype": null,
  "dataSessionOrigin": ""
}
```


Listing 4.2: Der dekodierte Wert von “string” in 4.1

Im Backend wird das Ereignis wieder dekodiert, fehlende Felder wie `datetime`, `apiVersion`, `userAgent` und `userId` hinzugefügt und anschließend als `Event` in BigQuery eingetragen.

4.1.3 Migration der Altdaten nach BigQuery

Wie bereits erwähnt existierte schon vor Beginn dieser Arbeit in rudimentärer Analytics-System, welches sowohl einzelne Klicks als auch neue Nutzer-Registrierungen unter dem Namen `Interaction` außerhalb von BigQuery abspeicherte. Dabei wurden über jede der Interaktionen folgende Informationen gespeichert:

```
public class Interaction {
  Long id;
  Date datetime;
  InteractionType interactionType;
  String elementId;
  String currentLocation;
  String randomVisitorID;
  String campaignId;
  String sessionOrigin;
  String userString;
}
```

Die meisten der hier vorhandenen Begriffe lassen sich auch im neuen Schema wiederfinden, wenn auch gegebenenfalls unter anderem Namen. Daher war eine Migration der Altdaten naheliegend. Da zu Testzwecken und um bei einem Wechsel der Systeme keine Daten zu verlieren, das neue und das alte System einige Wochen parallel liefen, musste hier lediglich darauf geachtet werden, keine bereits vom neuen System erfassten Interaktionen zu migrieren, um Duplikate zu vermeiden.

Die Migration selbst war davon abgesehen unkompliziert. So wurden alle vorhanden Interaktionen in Gruppen von je 20 parallel eingelesen, zu dem neuen Schema konvertiert, fehlende Felder so fern möglich abgeleitet und anschließend nach BigQuery geschrieben.

4.2 Visualisierung von Nutzerereignissen

4.2.1 Migration bisheriger Visualisierungen

Im alten Dashboard existierten vier Liniendiagramme die jeweils Anzahl der aktiven Benutzer, Anzahl der Besucher, Anzahl der Registrierungen und Anzahl aller Interaktionen über einen gewissen Zeitraum darstellten. Diese galt es zu erst zu migrieren.

Erstellen und Darstellen von Diagrammen

Als erster Schritt wurde eine neue Java-Klasse `TimeSeriesDiagram` implementiert, welche es erlaubt, im Backend Zeitreihendiagramme zu erstellen.

```
// Erstellen eines neuen Diagramms
TimeSeriesDiagram dia = new TimeSeriesDiagram();

// Beschriften der X-Achse
dia.defineLabels(
  Arrays.asList("20. 1.", "21. 1.", "22. 1.", "23. 1.")
);

// Hinzufügen von Datensätzen
dia.addDataset("Datensatz 1", Arrays.asList( 10,  14,  8,  11));
dia.addDataset("Datensatz 2", Arrays.asList(0.4, 4.2, 18, 400));
```

Listing 4.3: Beispielhafte Nutzung der TimeSeriesDiagram Klasse

Um in Zukunft noch weitere Diagrammtypen zu erlauben, wurde eine allgemeinere `Diagram` Klasse erstellt, von welcher die `TimeSeriesDiagram` Klasse erbt. Diese Oberklasse sorgt auch dafür, dass jedes Diagramm-Objekt, welches als Antwort auf eine HTTP Anfrage zurückgegeben wird, automatisch zu einem JSON-Objekt serialisiert wird.

```
{
  "type": "timeseries",
  "data": {
    "labels": [ "20. 1.", "21. 1.", "22. 1.", "23. 1." ],
    "sets": [
      {
        "name": "Datensatz 1",
        "data": [ 10, 14, 8, 11 ]
      },
      {
        "name": "Datensatz 2",
        "data": [ 0.4, 4.2, 18, 40 ]
      }
    ]
  }
}
```

Listing 4.4: Serialisiertes TimeSeriesDiagram

Im Frontend lässt sich aus diesem Objekt wiederum mithilfe einer neuen React-Komponente namens `DiagramsRenderer` der Graph zeichnen.

```
// Das serialisierte TimeSeriesDiagram aus dem vorherigen
   Beispiel
const exampleChartData = {"type": "timeseries", "data": { ... }}

// Die React-Komponente zum Darstellen des Diagramms
<DiagramsRenderer
  width={540}
  height={400}
  conf={exampleChartData}
  title="Mein Beispiel Graph"
/>
```

Listing 4.5: Darstellung von Diagrammen im Frontend

Dieser `DiagramsRenderer` wurde so implementiert, dass er Diagramme jeglicher Art entgegennehmen kann und diese erst zur Laufzeit anhand ihres Typen unterscheidet und korrekt darstellt. Der resultierende Code im Frontend ist so deutlich lesbarer und besser wartbar, da alle Details zum eigentlichen Rendern hinter einer Abstraktionsschicht versteckt sind und Diagramme, samt Diagrammtypen, komplett im Backend definiert werden.

Um das `TimeSeriesDiagram` aus dem vorherigen Beispiel darzustellen, verwendet der `DiagramsRenderer` die Bibliothek "Google Charts", was wie folgt aussieht:

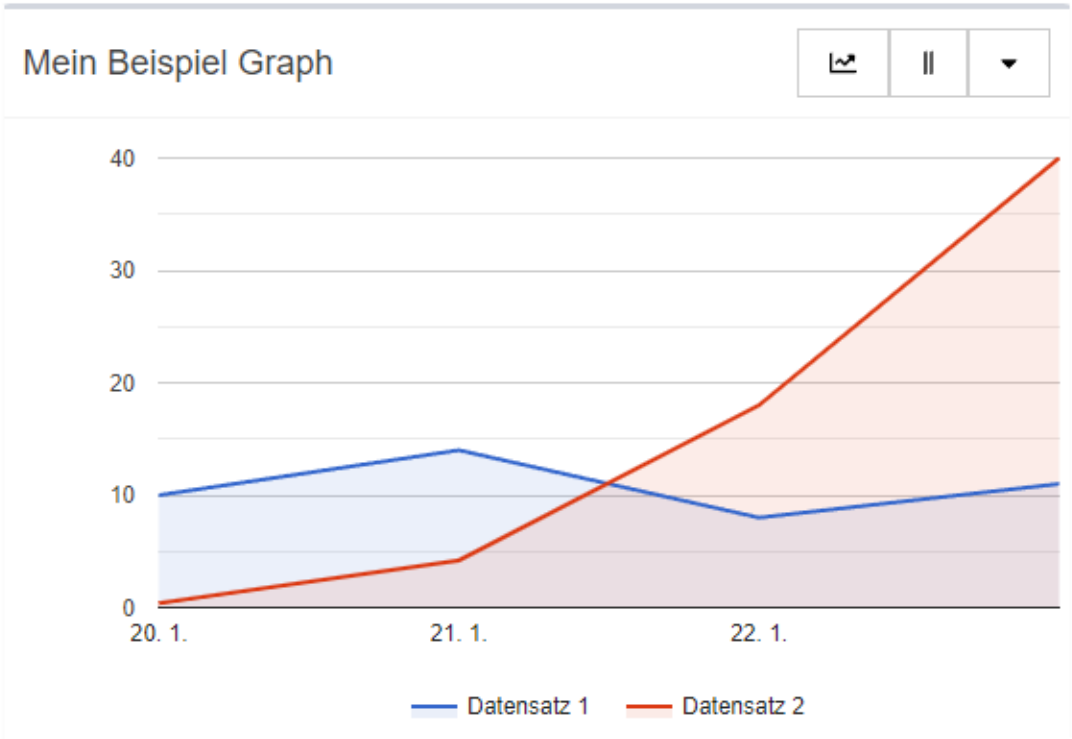


Abbildung 4.1: Beispieldiagramm wie es im Dashboard zu sehen ist

Datenbankanfragen

Um nun die korrekten Daten in ein eben vorgestelltes `TimeSeriesDiagram` einzufügen, müssen diese zunächst aus BigQuery geladen werden. Dazu wurde eine neue Java-Klasse erstellt, die den Namen `BigQueryRequestBuilder` trägt (im Folgenden auch nur "RequestBuilder" genannt) und einfache Datenbankzugriffe erlaubt.

```
BigQueryRequestBuilder request = ANALYTICS_CLIENT.newRequest()
  .select("userId, userVisitorId, datetime, dataCampaign")
  .where("type = @typ").withParameter("typ", eventType)
    .and("userId IS NOT null").or("userVisitorId IS NOT null")
    .and("userAgent IS null").or("NOT CONTAINS(userAgent,'bot')")
  .orderBy("datetime").desc()
  .limit(30);
```

Listing 4.6: Beispielhafte Nutzung der BigQueryRequestBuilder Klasse

Ein solches Interface hat viele Vorteile gegenüber manuell geschriebenen Datenbank-Abfragen.

Zunächst fällt auf, dass der Datenbankname nicht spezifiziert werden muss. Das ist einerseits vorteilhaft für die Wartbarkeit der Anfragenlogik, andererseits erlaubt es in automatisierten Tests auf anderen Datenbanken zu arbeiten.

Der RequestBuilder sorgt außerdem für sicherere Anfragen, indem beispielsweise Operatoren wie `LIMIT` Standardwerte erhalten, um nicht versehentlich gigantische Datenmengen anzufragen, und indem Nutzereingaben mithilfe von `withParameter()` hinzugefügt werden können um SQL-Injektion Angriffe zu verhindern.

Abschließend erlaubt es der RequestBuilder die Operanden der Anfrage nicht nur in beliebiger Reihenfolge hinzuzufügen, sondern auch mehrfach aufzurufen. Beim Erstellen der eigentlichen SQL-Anfrage werden dann mehrere Aufrufe desselben Operanden zusammengefügt. `AND` und `OR` Operatoren werden in konjunktiver Normalform angewendet.

Es wurde nun für jeden der zu migrierenden Graphen ein neuer Endpunkt erstellt, welcher wiederum mithilfe des RequestBuilders die passenden Daten aus BigQuery lädt und diese dann in ein `TimeSeriesDiagram` überführt. Mit Umstellung des Frontends auf diese neuen Endpunkte war die Migration abgeschlossen.

4.2.2 Filter

Filter bezeichnet im Folgenden eine Ansammlung an Einstellungen, um die auszuwertenden Daten einzugrenzen. Dazu zählen der betrachtete Zeitraum (letzten sieben Tage, letzter Monat, etc.), die Granularität der Daten (Ein Wert pro Tag,

pro Woche, etc.) und sogenannte Attributfilter, welche anschließend noch genauer definiert werden.

Frontend

Um die Auswertung der Daten zu vereinfachen und zu beschleunigen wurde beschlossen, die Filter global auf alle Graphen gleichzeitig anzuwenden und nicht einzeln pro Graph, wie es in vereinfachter Form vorher der Fall war.

Dazu wurde eine neue Leiste angebracht, welche Schaltflächen zum Konfigurieren der Filter beherbergt.

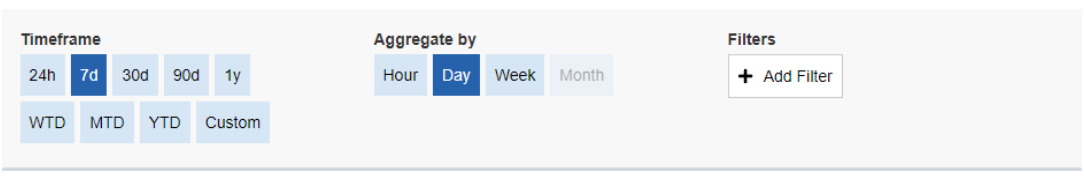


Abbildung 4.2: Die Filterleiste

Für schnelleres Umschalten wurden für häufig genutzte Zeitfenster eigene Knöpfe hinzugefügt, mit der Schaltfläche “Custom” lassen sich auch eigene Zeitrahmen definieren. Die Granularität passt sich beim Wechseln des Zeitfensters automatisch an. Einige Optionen werden hier jedoch eventuell deaktiviert, falls sie nicht zum gewählten Zeitraum passen.

Bei den Attributfiltern lassen sich beliebig viele Einträge hinzufügen. Jeder dieser besteht aus einem Namen und einem Wert. Als Namen sind die meisten Attribute der Ereignisse wählbar. Als Wert lässt sich je nach Attributname aus einer dynamisch geladenen Liste wählen oder Freitext eingeben.

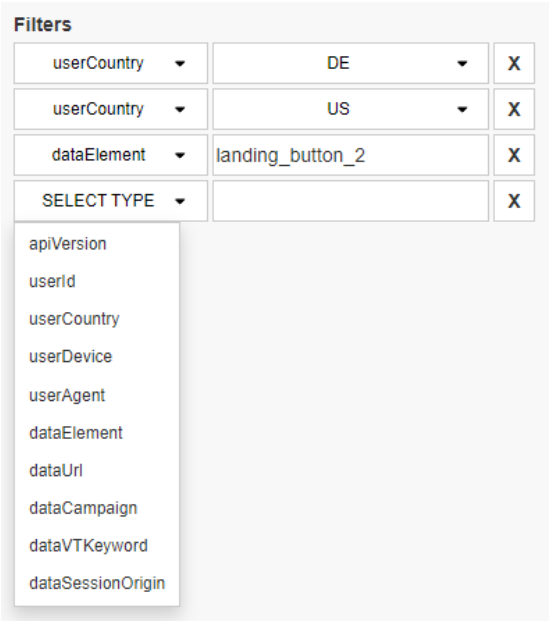


Abbildung 4.3: Anpassung der Attributfilter

Fügt man unterschiedliche Attribute hinzu, muss ein Ereignis alle erfüllen, um Teil der Ergebnismenge dieser Anfrage zu sein. Fügt man hingegen dasselbe Attribut mehrfach hinzu, werden die einzelnen Werte mittels `OR` verbunden und es muss nur eines davon übereinstimmen. In 4.3 müssen Ereignisse beispielsweise entweder aus `DE` oder `US` kommen, in jedem Fall jedoch das Element `landing_button_2` referenzieren.

Möchte man alle Werte bis auf Einen zulassen, so kann diesem ein `!` vorangestellt werden, um die Anfrage zu negieren. Möchte man nur den Teil eines Wortes suchen, so lässt sich der Platzhalter `*` einsetzten, der beliebig viele Buchstaben an seiner Stelle zulässt. Um eine `AND`-Verbindung desselben Attributs zu erzwingen, sinnvoll beispielsweise bei negierten Anfragen, können mehrere Begriffe mit einem Komma getrennt werden, statt zusätzliche Filter hinzuzufügen.

Nach Anpassen der Filter werden alle Diagramme aktualisiert. Dabei werden die Filter in der jeweiligen Anfrage mit an das Backend geschickt.

Backend

Im Backend werden die Filter automatisch aus der Anfrage genommen und in ein `AnalyticsInsightsFilters` Objekt verwandelt. Dieses wiederum implementiert das `BigQueryRequestTransformer` Interface, was folgendes Programmiermuster erlaubt:

```
public Diagram eventsOverTime(AnalyticsInsightsFilters filters) {
  BigQueryRequestBuilder req = ANALYTICS_CLIENT.newRequest()
    .apply(filters) // Anwenden der Filter in einer Zeile Code
    .select("COUNT(*) AS count")
    .limit(1000);

  // ...
}
```

Listing 4.7: Anwenden der Filter auf Server-Seite

Der `BigQueryRequestBuilder` besitzt neben den normalen SQL-Operanden auch eine Methode namens `apply()`, welche einen `BigQueryRequestTransformer` entgegennimmt. Der wiederum kann so selbst auf den RequestBuilder zugreifen und eigene SQL-Operationen hinzufügen, da der `BigQueryRequestBuilder` diese in beliebiger Reihenfolge erlaubt. Das Resultat ist Vermeidung von Code-Duplizierung, sowie einfache Wartbarkeit und Erweiterbarkeit.

4.2.3 Mehrdimensionale Visualisierungen

Wie bereits in den Anforderungen festgehalten, sollte es ebenfalls möglich gemacht werden, verschiedene Metriken im selben Graphen miteinander zu vergleichen. Dazu wurde der Filterleiste eine neue Option hinzugefügt, die es erlaubt ein beliebiges Ereignisattribut zu wählen, anhand welchem differenziert werden soll.

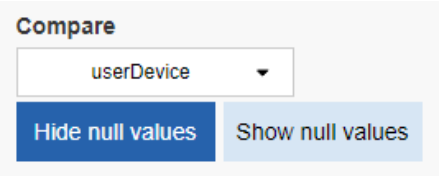


Abbildung 4.4: Anpassung des Vergleichswertes

Ist hier ein Attribut ausgewählt, wird beim Auswerten der Daten nun für jeden individuellen Attributwert ein eigener Datensatz geführt, welche alle anschließend im selben Graphen dargestellt werden.

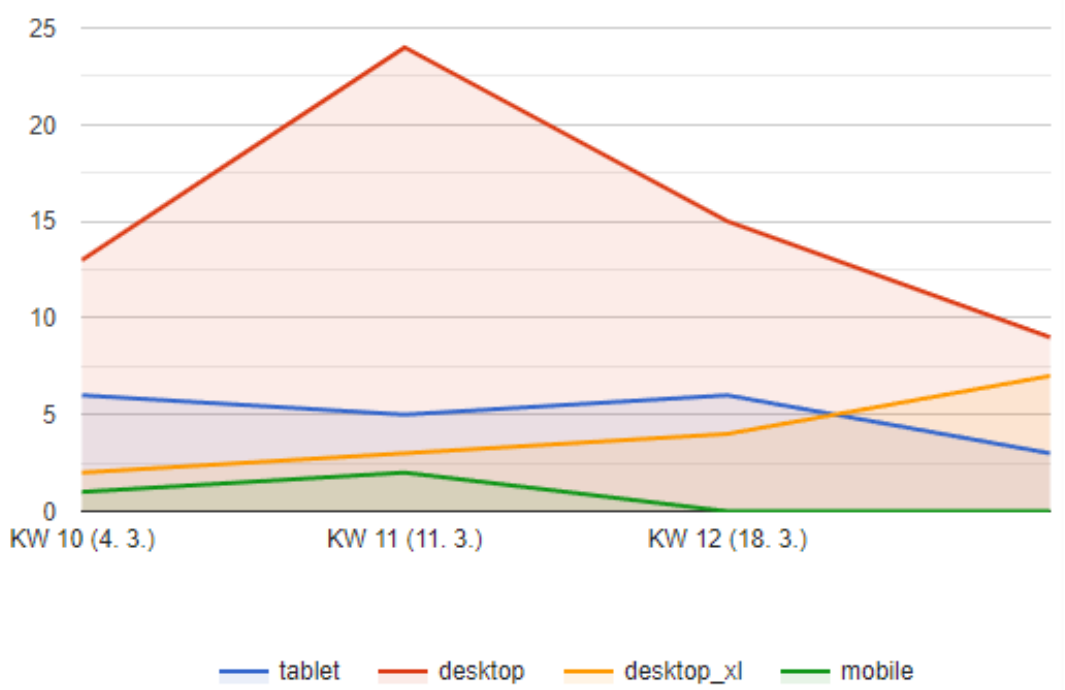


Abbildung 4.5: Graph der nach "userDevice" aufgeteilt wurde

Möchte man nicht für alle Werte einen eigenen Datensatz sehen, lassen sich diese weiterhin mit Filtern eingrenzen.

Die serverseitige Implementierung lief genau wie die restlichen Filter über das `AnalyticsInsightsFilters` Objekt ab, welches per `apply()` auf die Query angewendet wird. So mussten keine Endpunkte angepasst oder umgeschrieben werden, um mehrdimensionale Visualisierungen zu unterstützen.

4.2.4 Weitere Anzeigeoptionen

Mit mehreren Datensätzen in einem Graphen kann es schnell unübersichtlich werden. Daher wurden zwei Anzeigeoptionen hinzugefügt, welche pro Graph individuell umgeschaltet werden können. Die Erste erlaubt es, zwischen Linien- und Balkendiagramm umzuschalten, die Zweite ändert, ob die einzelnen Datensätze individuell oder gestapelt angezeigt werden. Beide Optionen sind unabhängig voneinander umzuschalten.

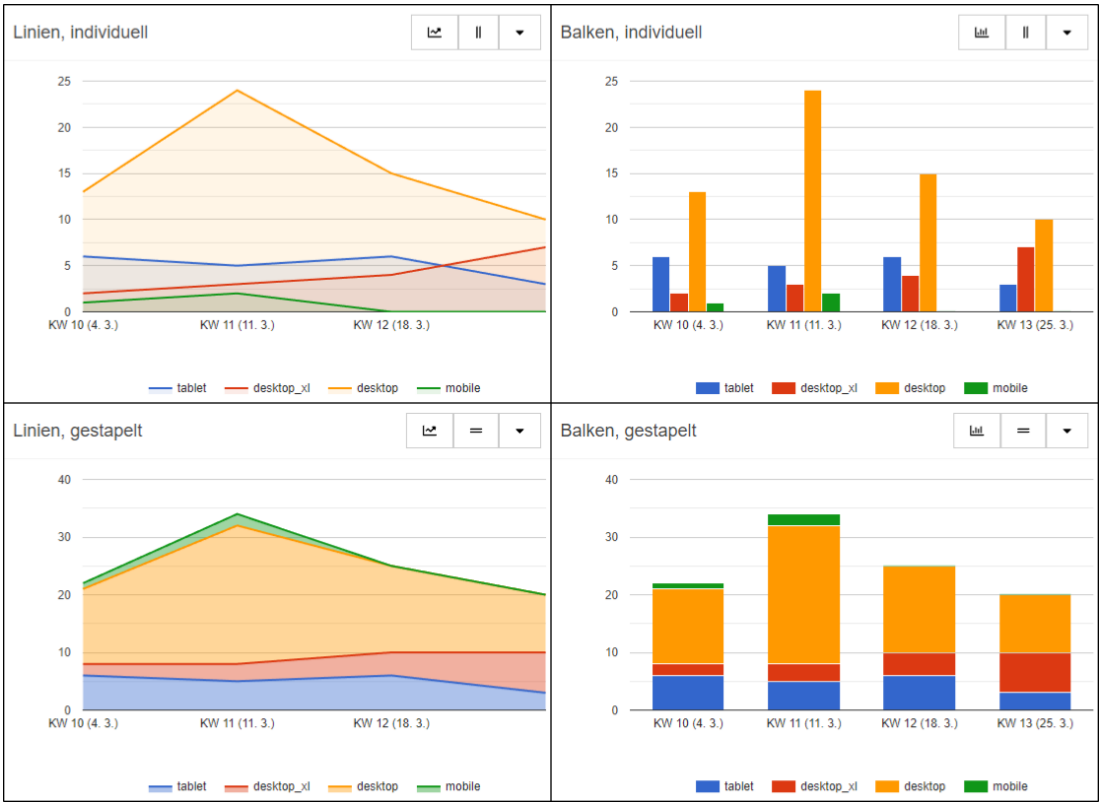


Abbildung 4.6: Übersicht über die Anzeigeoptionen

Ein weiterer Knopf erlaubt das Ausblenden einzelner Graphen für eine bessere Übersicht.

4.2.5 Neue Visualisierungen

Um besser zu verstehen, wie Nutzer durch die Webseite navigieren, wurden neue Visualisierungen benötigt. Sogenannte "Sankey Diagramme" sind eine Unterklasse der Flussdiagramme und werden zur Modellierung von Zustandsübergängen in Mengen verwendet (Eurostat, 2023). Als eine solche Menge lässt sich auch die Besucherschaft von QDAcity betrachten. Als Zustände wiederum können hier beispielsweise die URLs, welche besucht werden, fungieren.

So lässt sich zum Beispiel folgendes Diagramm modellieren, bei welchem alle Seitenwechsel den Zustandsübergängen entsprechen.

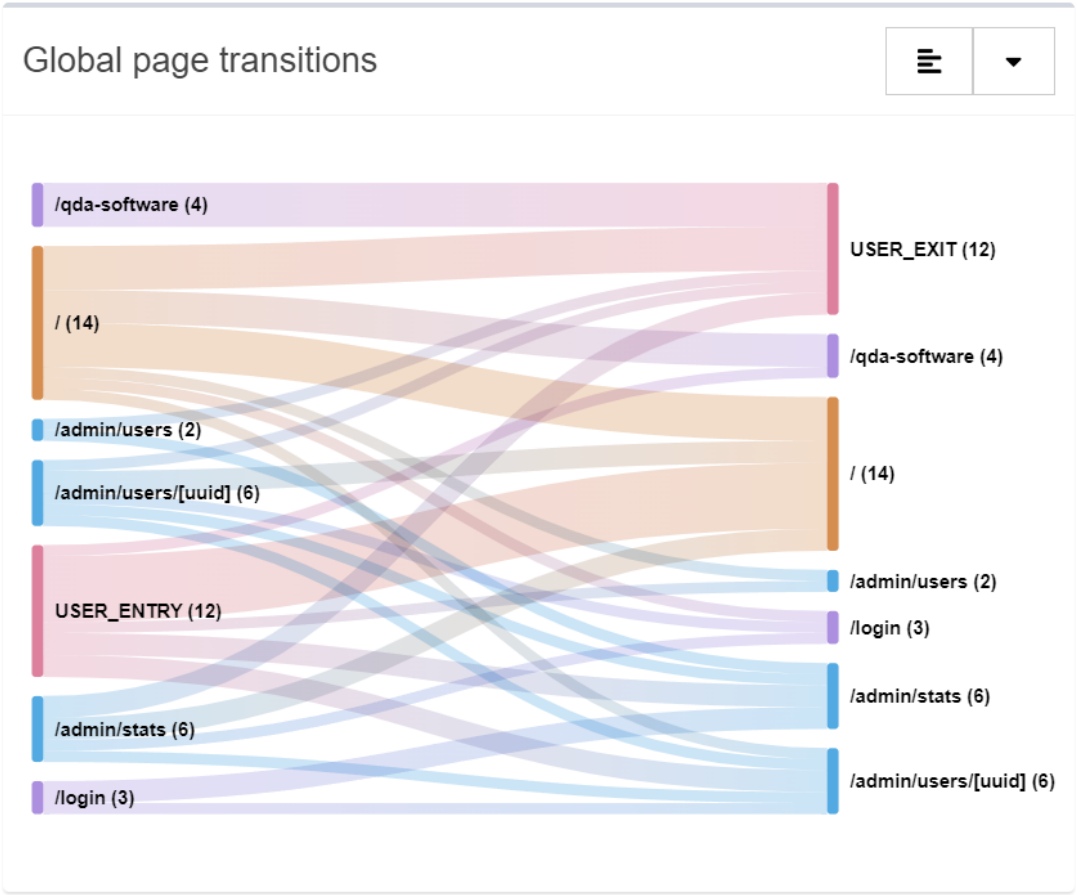


Abbildung 4.7: Globale Seitenwechsel als Sankey Diagramm

Das Diagramm erlaubt einen schnellen Überblick darüber, von welchen Seiten aus häufig beziehungsweise selten auf welche anderen Seiten gewechselt wird.

Die Implementierung der Sankey Diagramme baut auf den bereits etablierten Grundstrukturen auf. Im Backend erbt die dafür erstellte `SankeyDiagram` Klasse, genauso wie die Zeitreihendiagramme, von der `Diagram` Oberklasse. So lässt sich die Ereignismenge, die für das Zeichnen des Diagramms verwendet wird, analog zu anderen Diagrammtypen durch die bereits beschriebenen Filter weiter eingrenzen, ohne dass im Backend weitere Implementierungsarbeiten nötig waren. Im Frontend werden die Diagramme dann in einer eigens gebauten Komponente mithilfe des Open-Source Frameworks d3.js dargestellt.

Durch die Einhaltung dieser einheitlichen Schnittstellen lassen sich Sankey Diagramme auch zukünftig ohne großen Aufwand in anderen Bereichen von QDAcity wiederverwenden.

Die Implementierung beinhaltet zusätzliche Optionen für einen einfacheren Umgang mit dem Diagrammtypen. So kann die Farbkodierung basierend auf den Namen der Mengen definiert werden, um beispielsweise zusammenhängende Mengen in derselben Farbe darzustellen. Ebenso können verschiedene Arten von Filtern angewendet werden, um etwa kleine Mengen auszublenden oder zusammenzuführen, da diese bei großen Diagrammen häufig die Lesbarkeit verschlechtern.

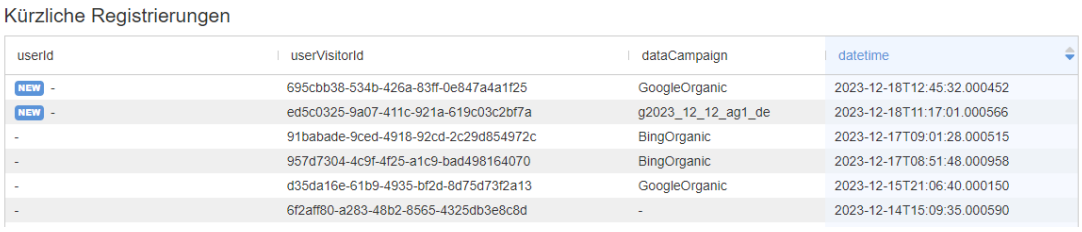
4.3 Visualisierung einzelner Nutzer und Navigationspfade

Alle bisher diskutierten Visualisierungen befassen sich mit der gesamten oder einer gezielten Teilmenge aller Nutzer. Je nach Fragestellung oder Situation kann es allerdings von Interesse sein, einzelne Nutzer individuell anzuzeigen, um deren "User Journey", also Reise durch die Webseite, besser zu verstehen. Da alle Ereignisse die `userId` oder `userVisitorId` aufzeichnen, sowie die `userSession`, lassen sich alle Ereignisse für einen individuellen Nutzer gruppieren.

4.3.1 Übersicht kürzlich aktiver Nutzer

Als Einstiegspunkt wurde dem Dashboard eine neue Seite hinzugefügt, welche zwei Listen mit Nutzern zeigt. Die erste enthält die 30 zuletzt registrierten Benutzer, die zweite die 30 letzten Besucher, egal ob registriert oder nicht. Mit einem Klick auf einen dieser Listeneinträge gelangt man dann auf eine neue Seite mit Details zu der Person.

In beiden Listen sind neben der Nutzer-ID auch das Datum und die dazugehörige Kampagne aufgezeigt. Nachdem die Nutzer in diesen Listen allerdings nicht immer angemeldet sind und so häufig keinen Klarnamen, sondern lediglich eine anonyme UUID als eindeutiges Identifikationsmerkmal tragen, musste eine Funktion her, um nicht den Überblick zu verlieren. So bekommen alle neuen Erscheinungen in den Listen eine Notiz mit dem Text `NEW`, welche so lange vor der Nutzer-ID steht, bis der Benutzer angeklickt wurde.


Kürzliche Registrierungen

userId	userVisitorId	dataCampaign	datetime
NEW -	695cbb38-534b-426a-83ff-0e847a4a1f25	GoogleOrganic	2023-12-18T12:45:32.000452
NEW -	ed5c0325-9a07-411c-921a-619c03c2bf7a	g2023_12_12_ag1_de	2023-12-18T11:17:01.000566
-	91babade-9ced-4918-92cd-2c29d854972c	BingOrganic	2023-12-17T09:01:28.000515
-	957d7304-4c9f-4f25-a1c9-bad498164070	BingOrganic	2023-12-17T08:51:48.000958
-	d35da16e-61b9-4935-bf2d-8d75d73f2a13	GoogleOrganic	2023-12-15T21:06:40.000150
-	6f2aff80-a283-48b2-8565-4325db3e8c8d	-	2023-12-14T15:09:35.000590

Abbildung 4.8: Liste der kürzlichen Registrierungen

4.3.2 Nutzerdetailansicht

Auf der Übersichtsseite werden nun alle Ereignisse, die über den Nutzer zu finden sind zusammengeführt. Dies kann gegebenenfalls zur Herausforderung werden, da in manchen Fällen Nutzer über mehrere Sitzungen hinweg erkannt werden müssen, ohne dass sich diese auf der Seite angemeldet haben. Abhilfe schafft hier die `userVisitorId`, welche uns erlaubt auf Fingerprinting (Microsoft, 2024) oder ähnliche Praktiken zu verzichten.

Sobald alle Ereignisse gesammelt wurden, kann die Auswertung beginnen. Dabei werden zunächst einige Kerninformationen aus den einzelnen Ereignissen aggregiert. Dazu zählen die Gesamtzahl der Sitzungen, der benutzte Gerätetyp und das Herkunftsland. Unter "Funnel", zu Deutsch Trichter, ist zu sehen, über welchen Weg der Nutzer die Plattform gefunden hat. Eine zeitliche Einordnung wichtiger Ereignisse schließt diese Übersicht ab.

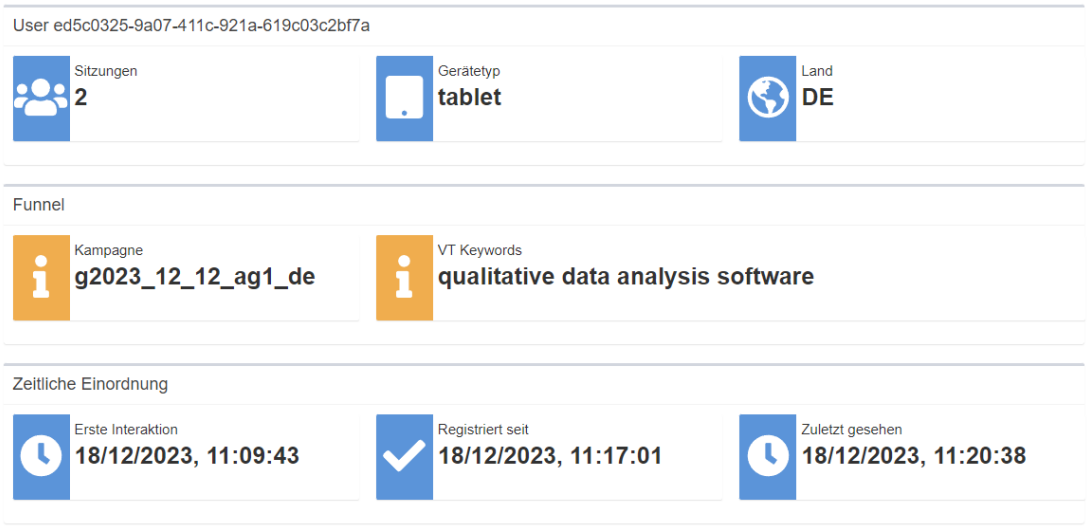


Abbildung 4.9: Kerninformationen über einen Benutzer

Im Anschluss folgt eine Auflistung aller Sitzungen und jeweiliger Ereignisse, für den Fall, dass eine präzisere Analyse geplant ist.

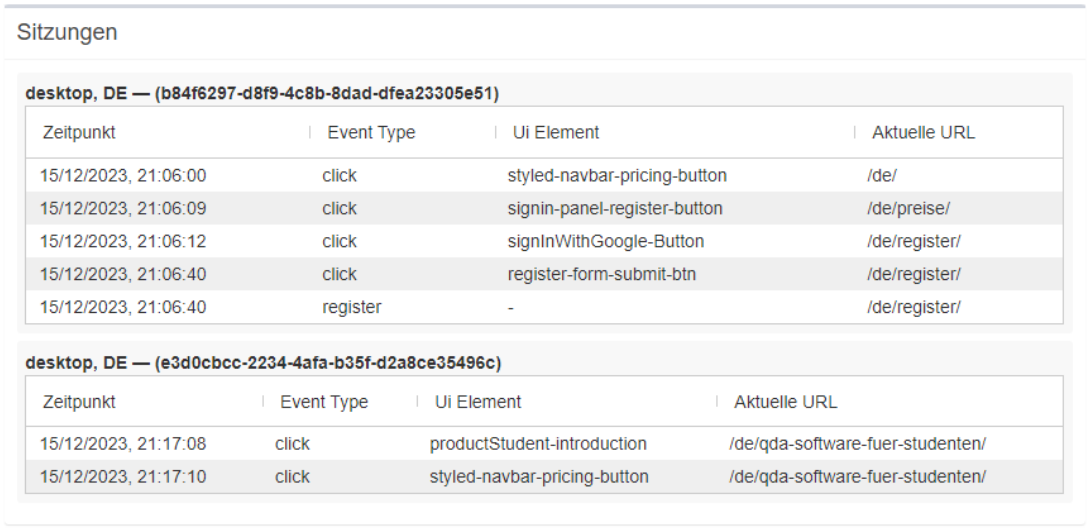


Abbildung 4.10: Sitzungsübersicht über einen Benutzer

Eine häufige Fragestellung bei der Auswertung einzelner Nutzerprofile ist, welchen Weg die Nutzer durch die Webseite nehmen. Also welche Seiten in welcher Reihenfolge besucht werden. So wurde hierfür eine auf Sankey Diagrammen basierende Visualisierung hinzugefügt, welche die Nutzerpfade übersichtlich und schnell zugänglich darstellt.

Abbildung 4.11: Beispielhafter Weg eines Nutzers

Der Nutzer in 4.11 hatte zwei Sitzungen, eine davon begonnen auf der Seite `/qda-software`, die andere auf der Startseite `/`. Beide Sitzungen führten über die `/preise` Seite. Hier endete eine Sitzung, die andere führte weiter zur `/register` Seite.

Um diese Darstellung zu erlauben werden alle Sitzungen der jeweiligen Person zuerst als Liste von Seitenwechseln betrachtet. Als Sankey Diagramm würde dies dann mit den Daten aus 4.11 wie folgt aussehen:

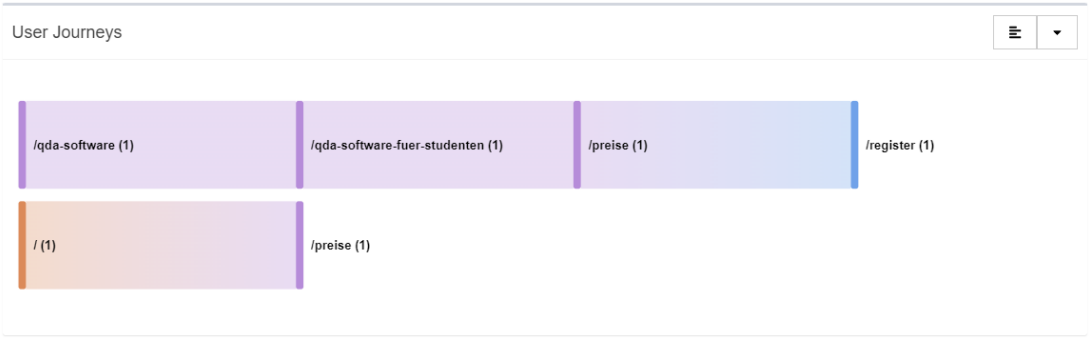


Abbildung 4.12: Nutzerpfade ohne Ausrichtung

In einem zweiten Schritt werden dann die einzelnen Sitzungen noch so ausgerichtet, dass sich häufig auftretende Muster im Graph überschneiden.

Wechselt beispielsweise ein Nutzer in mehreren Sitzungen von der Startseite auf die Login-Seite, sollen diese im Diagramm zusammengefasst dargestellt werden.

Abbildung 4.13: Weiterer Nutzerpfad mit vier Sitzungen

In allen Nutzerpfad-Diagrammen werden der Übersicht halber kontextuell verwandte Seiten in derselben Farbe dargestellt.

5 Auswertung

Die Umsetzung des neuen Analytics-Warehouse war ein großer Erfolg. Das alte Dashboard wurde im vollen Umfang abgelöst und intuitiver gestaltet. Die Datensammlung wurde zudem nicht nur umfangreicher, sondern auch verlässlicher, da sowohl auf Drittanbieter verzichtet wurde, als auch explizite Maßnahmen wahrgenommen wurden, um die Funktion des Systems auch bei Besuchern mit vielen Browsererweiterungen zu ermöglichen.

Die neuen Filtermöglichkeiten im Dashboard können für präzisere Auswertungen genutzt werden und haben bereits in wenigen Wochen nach Inbetriebnahme die Effektivität diverser Werbekampagnen dargelegt und so bei der Entscheidungsfindung von Folge-Kampagnen geholfen. Durch neue Informationen wie dem Gerätetypen oder der VT-Werte in Kombination mit der Möglichkeit im selben Graphen verschiedene Attributwerte in unterschiedliche Datensätze aufzuteilen, konnte die eigene Besucherschaft genauer verstanden werden. Das wiederum hat nicht nur Entscheidungen bezüglich neuer Werbekampagnen erleichtert, sondern hilft dem QDAcity-Team ganz allgemein das eigene Produkt zu verbessern und Prioritäten bei der Weiterentwicklung der Anwendung zu setzten.

Die Detailansicht für einzelne Nutzer erlaubt zusätzliche Einblicke in einzelne Sitzungen für ein exaktes Verständnis der verschiedenen Personengruppen, die die Seite benutzen. Rohdaten sind außerdem einfach über BigQuery zugänglich und können für erweiterte Datenanalyse auch exportiert und mit externer Software ausgewertet, sowie mit anderen Datenquellen kombiniert werden.

Aufgrund zeitlicher Einschränkungen mussten jedoch leider bei den Plänen zum individuellen Anpassen des Dashboards auf einzelne Analytiker Kompromisse eingegangen werden. So existieren bisher keine editierbaren Profile und auch nicht die Möglichkeit, neue Graphen visuell hinzuzufügen. Durch die einfach zu bedienenden Schnittstellen und klar dokumentierten Komponenten in Front- und Backend können neue Graphen jedoch mit geringem Aufwand implementiert werden.

6 Ausblick

An erster Stelle stehen hier selbstverständlich die bereits in den Anforderungen beschriebenen, aber noch ausstehenden Ziele. Individuelle Profile und ein visueller Grapheditor würden das Dashboard noch abrunden.

Auch sinnvoll wäre es, zukünftig Benachrichtigungen bei besonderen Ereignissen einzurichten. Sollten beispielsweise plötzlich viele Personen die Seite besuchen erfordert dies gegebenenfalls die Aufmerksamkeit eines QDAcity Mitarbeiters. Das in dieser Arbeit implementierte Warehouse sammelt bereits alle Informationen, die für ein solches System notwendig wären.

Auch denkbar wäre eine autonome Mustererkennung in den gesammelten Ereignissen, wie beispielsweise in Nguyen et al. (2017). So könnte man zum Beispiel Sitzungen erkennen, in denen der Nutzer zwar die Registrierungsseite besucht, sich jedoch nicht registriert hat. Solche aggregierten Werte wären gerade in zeitlichem Verlauf äußerst interessant und aktuell nur mit großem manuellem Aufwand möglich.

Abschließend wäre eine Verknüpfung mit weiteren externen Daten gegebenenfalls Vorteilhaft. Andere Produkte der GCP können bereits ihre Daten nach BigQuery schreiben, darunter unter anderem die "Google Search Console", welche Informationen zum eigenen Ranking in Suchergebnissen sammelt (Google, n. d. a). Mit mehr Daten ließen sich noch weitere Schlüsse ziehen.

Literaturverzeichnis

Bekavac, I., & Praničević, D. G. (2015). Web analytics tools and web metrics tools: An overview and comparative analysis. *Croatian Operational Research Review*, *6*(2). https://doi.org/10.17535/crorr.2015.0029

Cloudflare. (n. d.). *What is a web crawler? | How web spiders work.* Verfügbar 2. April 2024 unter https://www.cloudflare.com/en-gb/learning/bots/what-is-a-web-crawler/

Eurostat. (2023). *Sankey diagrams for energy balance.* Verfügbar 2. April 2024 unter https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Sankey_diagrams_for_energy_balance

Google. (n. d. a). *About Search Console.* Verfügbar 2. April 2024 unter https://support.google.com/webmasters/answer/9128668?hl=en

Google. (n. d. b). *Set up tracking with ValueTrack parameters.* Verfügbar 2. April 2024 unter https://support.google.com/google-ads/answer/6305348

Google. (2024a). *BigQuery: Cloud data warehouse to power your data-driven innovation.* Verfügbar 2. April 2024 unter https://cloud.google.com/bigquery

Google. (2024b). *Datastore: Datastore is a highly scalable NoSQL database for your web and mobile applications.* Verfügbar 2. April 2024 unter https://cloud.google.com/datastore

Google. (2024c). *Search Engine Optimization (SEO) Starter Guide.* Verfügbar 1. April 2024 unter https://developers.google.com/search/docs/fundamentals/seo-starter-guide

Google Cloud. (2020). *Spotify: The future of audio. Putting data to work, one listener at a time.* Verfügbar 2. April 2024 unter https://cloud.google.com/customers/spotify

Leach, P. J., Salz, R., & Mealling, M. H. (2005, Juli). A Universally Unique IDentifier (UUID) URN Namespace. https://doi.org/10.17487/RFC4122

Mandarina & Hase, M. (2019). *Was ist Vendor Lock-in?* Verfügbar 2. April 2024 unter https://www.it-business.de/was-ist-vendor-lock-in-a-879691/

Microsoft. (2024). *Overview of device fingerprinting.* Verfügbar 2. April 2024 unter https://learn.microsoft.com/en-us/dynamics365/fraud-protection/device-fingerprinting

Mozilla. (2023a). *Document: referrer property*. Verfügbar 2. April 2024 unter https://developer.mozilla.org/en-US/docs/Web/API/Document/referrer

Mozilla. (2023b). *Geolocation API*. Verfügbar 2. April 2024 unter https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API

Mozilla. (2023c). *User-Agent*. Verfügbar 2. April 2024 unter https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent

Mozilla. (2023d). *Window: sessionStorage property*. Verfügbar 2. April 2024 unter https://developer.mozilla.org/en-US/docs/Web/API/Window/sessionStorage

Nguyen, P. H., Turkay, C., Andrienko, G., Andrienko, N., & Thonnard, O. (2017). A Visual Analytics Approach for User Behaviour Understanding through Action Sequence Analysis. In M. Sedlmair & C. Tominski (Hrsg.), *EuroVis Workshop on Visual Analytics (EuroVA)*. The Eurographics Association. https://doi.org/10.2312/eurova.20171122

Podojil, E. (2021). *The evolution of data architecture at The New York Times*. Verfügbar 2. April 2024 unter https://cloud.google.com/blog/products/data-analytics/how-the-new-york-times-build-an-end-to-end-cloud-data-platform

Red Hat. (2020). *What is a REST API?* Verfügbar 2. April 2024 unter https://www.redhat.com/en/topics/api/what-is-a-rest-api

Salesforce. (n. d.). *What is SaaS?* Verfügbar 3. April 2024 unter https://www.salesforce.com/eu/learning-centre/tech/saas/

Simic, M. (2024). *Largest Contentful Paint (LCP)*. Verfügbar 2. April 2024 unter https://www.baeldung.com/cs/missing-vs-sparse-data

Spellman, J., Bradley, R., Cooper, M., Lauriat, S., Adams, C., & Campbell, A. (2023). *W3C Accessibility Guidelines (WCAG) 3.0*. Verfügbar 2. April 2024 unter https://www.w3.org/TR/wcag-3.0/

Springer. (n. d.). *Qualitative Datenanalyse*. Verfügbar 1. April 2024 unter https://lehrbuch-psychologie.springer.com/glossar/qualitative-datenanalyse

Waisberg, D., & Kaushik, A. (2009). Web Analytics 2.0: Empowering Customer Centricity. *Croatian Operational Research Review*. Verfügbar 3. April 2024 unter https://docplayer.net/7409731-Web-analytics-2-0-empowering-customer-centricity.html

Walton, P., & Pollard, B. (2024). *Largest Contentful Paint (LCP)*. Verfügbar 2. April 2024 unter https://web.dev/articles/lcp