# Dokumentation der Softwarearchitektur des Projekts Twitter-Dash

Hahn, Bastian Kleber, Martin Klier, Andreas Kreussel, Lukas Paris, Felix Ziegler, Andreas
*b.hahn@oth-aw.de m.kleber2@oth-aw.de a.klier@oth-aw.de l.kreussel@oth-aw.de f.paris1@oth-aw.de a.ziegler1@oth-aw.de*

***Abstract*—In diesem technischem Report wird die Softwarearchitektur des Projekts Twitter-Dash vorgestellt. Das Projekt wurde im Rahmen der Vorlesung Big Data and Cloud Computing (bdcc) implementiert. Ziel der Implementierung ist es, ein Dashboard zu erstellen, dass aktuelle und historische Informationen zu Twitter Trends anzeigen und analysieren kann.**

***Index Terms*—Twitter, Big Data, Database, Web UI, Data Analysis**

## I. Introduction

Bei der Applikation Twitter-Dash werden von der Twitter API alle 15 Min die aktuellen Trends abgegriffen und in einer lokalen Datenbank gespeichert. Die Kommunikation zwischen den einzelnen Services ist über eine gRPC Schnittstelle realisiert. Die Interaktion des Anwenders mit der Applikation erfolgt dabei über eine Web-UI basierend auf dem React Web-Framework.

## II. Architektur Allgemein

Als Architektur wird ein Ansatz verfolgt, der sich stark an Microservices orientiert. Dadurch wird eine abgeschlossene Logik von Front- und Backendkomponenten erreicht, die jeweils von einem Subteam entwickelt wird. Diese gekapselten Einheiten können jeweils als isolierte Applikation der Containervisualisierung Docker betrieben werden. Durch eine zentrale Konfigurationsdatei des Containers können diese zwischen den Teammitgliedern ausgetauscht und ausgeführt werden. Damit wird eine hohe Portabilität gewährleistet. Die Kommunikation der Teilsysteme wird durch eine sprachunabhängige gRPC-Schnittstelle angeboten. Dadurch wird es möglich einzelne Komponenten des Gesamtsystems auszutauschen, ohne größere Veränderungen an den anderen Microservices vornehmen zu müssen [1].

## III. Datenakquise

Um Daten für das Dashboard zu erhalten, werden zwei unterschiedliche Vorgehensweisen verwendet. Der verbreiteste Ansatz ist die Abfrage über die offizielle Twitter API [2]. Über diese werden die aktuellen Hashtags periodisch abgerufen. Außerdem wurde Twint, ein fortgeschrittenes Scraping-Tool verwendet. Dieses Tool liefert vollständige Tweets, inklusive zugehöriger Meta-Daten, ohne direkten Zugriff auf die Twitter API zu benötigen [3].

### *Trend-Service*

Für den Zugriff auf die Twitter-API wurde die Python-Bibliothek Tweepy [4] verwendet. Diese Bibliothek bietet einen Client zur Verbindung zur Twitter-API. Über diesen Client werden alle 15 Minuten die aktuellen Trends pro Land abgefragt und nach Hashtags gefiltert. Diese werden über den Conductor zur Speicherung an die Datenbank gesendet.

Der Trend-Service bietet folgende Funktionen:

- getAvailableTrends:
  Liefert alle Länder zurück, für die die Twitter API aktuell Trends zur Verfügung stellt.
- getTrending:
  Liefert die aktuellen Hashtags in einer Liste zurück. Die Listenelemente umfassen:
  - Hashtag
  - Platzierung nach Aktualität im Land
  - Anzahl der Tweets zu diesem Hashtag
  - Zeitstempel der Abfrage
  - Land
- getTweetCount:
  Liefert die Anzahl der Tweets in einem vorgegebenen Zeitraum zu einem Hashtag zurück.

### *Twint-Service*

Da die Nutzung der Twitter-API eingeschränkt ist, wurde Twint verwendet, um viele Tweets zu laden. Der Twint-Service wird nur auf Anfrage des Conductors ausgeführt. Dieser gibt einen Hashtag, einen Start- und Endzeitpunktsangabe und eine Sprache vor. Zu dieser Anfrage werden Tweets mit Twint gesammelt und nach Bereinigung von unwichtigen Metadaten zurück an den Conductor gegeben.

Die zurückgegebenen Elemente enthalten die folgenden Informationen:

- Tweet-ID
- Conversation-ID
- Zeitstempel
- Textinhalt des Tweets
- Anzahl der Likes
- Anzahl der Retweets
- Hashtags in dem Tweet
- Sprache

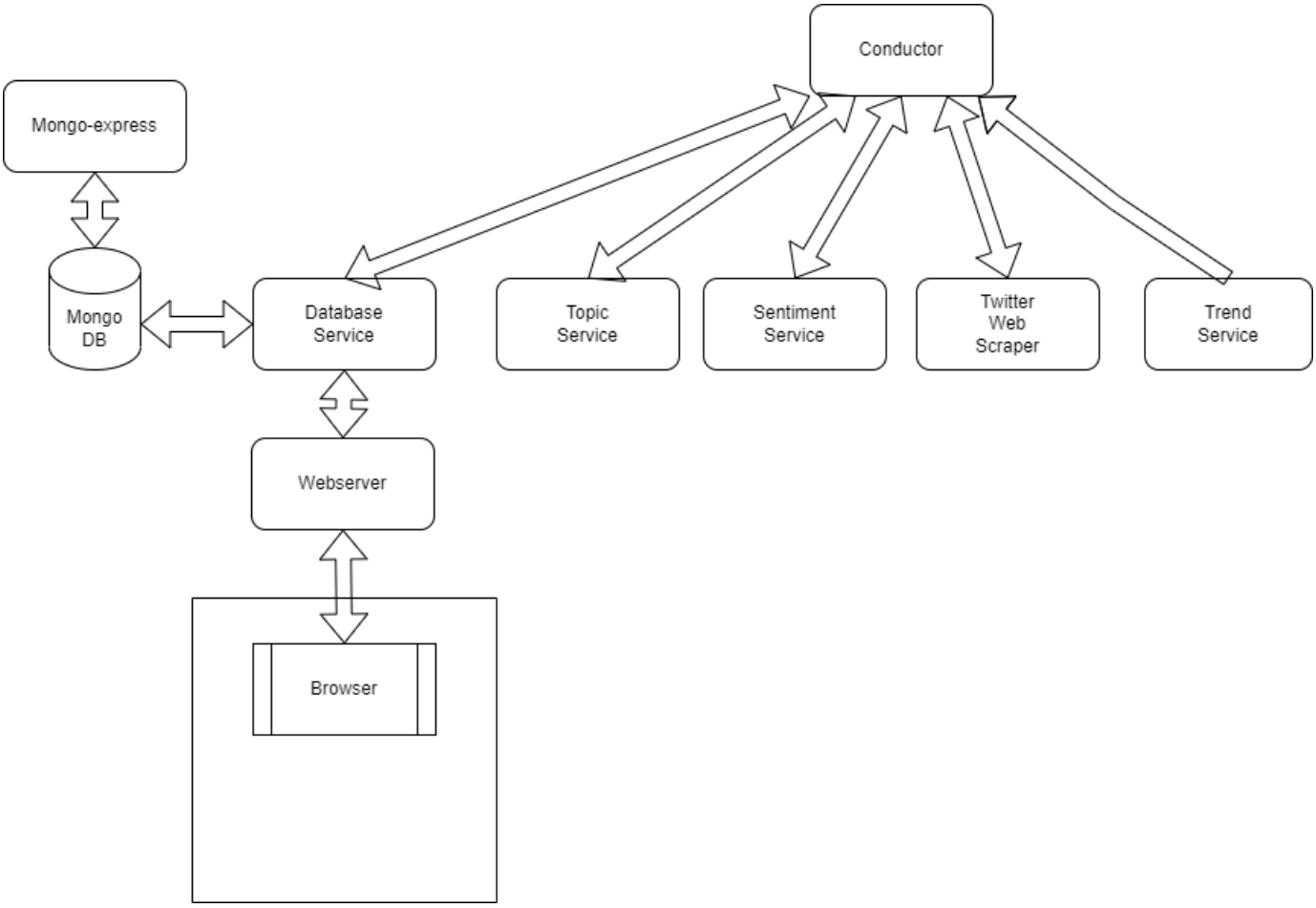


Fig. 1. Übersicht Gesamtsystem

## IV. Datenverarbeitung

Aus den gesammelten Daten können weitergehende Informationen extrahiert werden. Hier wurde ein Service implementiert, der das Sentiment eines Tweets analysiert.

*Sentiment-Analyse*

Bei der Sentimentanalyse werden die in einem Text ausgedrückten Meinungen mithilfe von NLP analysiert, um festzustellen, wie die Einstellung des Verfassers zu einem bestimmten Thema ist. Zur Analyse der Stimmungen wurden zwei verschiedene Modelle eingebunden, ein bereits vortrainiertes neuronales Sprachmodell (BERT [5]) und ein ein stochastisches Modell (TextBlob [6]). Aufgrund des hohen Ressourcenverbrauchs des Tranformermodells (BERT) wurde das TextBlob-Modell für die Ausführung gewählt.

Die Ausführung der Sentiment-Analyse erfolgt über einen Service, der auf Anfrage des Conductors basiert. Bei dieser Anfrage wird ein Text übergeben, der analysiert werden soll, sowie eine Sprache, die zur Wahl des richtigen TextBlob-Modells benötigt wird. Bevor der Text analysiert werden kann muss er bereinigt werden. Dabei werden zum Beispiel unerwünschte Zeichen entfernt und die Zeichenkette in Kleinbuchstaben umgewandelt. Als Ausgabe liefert der Service eine Zahlenwert zwischen -1 und 1. Dabei wird der Wert -1 für negative und 1 für positive Stimmungen angenommen.

## V. Datenbank

Alle Datenbankanfragen werden durch einen database service abstrahiert, der die Funktionen als gRPC Schnittstelle anbietet. Dabei laufen die Datenbank und der database service in jeweils einem eigenen Container. Durch die gRPC Schnittstelle können die Daten typsicher und Platform- sowie Programmiersprachen-unabhängig an den database service übergeben und von diesem zurückgeliefert werden.
Vom database service werden folgende Funktionen angeboten:
Zum Speichern in die Datenbank:
**Trends speichern (StoreTrends)**

- Beschreibung: Speichert Trends in die Datenbank
- Parameter:
  - Zeitstempel
  - Liste von Trends mit folgenden Attributen:
    - trendtype

        * name
        * country
        * placement
        * tweetVolume24
- Rückgabe: Keine

**Sentiment speichern (StoreSentiment)**
- Beschreibung: Speichert Sentiment in die Datenbank
- Parameter:
  - Liste von Sentiments mit folgenden Attributen:
    * tweet
    * Sentiment
    * topic
- Rückgabe: Keine

zum Laden aus der Datenbank:

**aktuelle Trends abrufen (GetCurrentTrends)**
- Beschreibung: Lädt aktuelle Trends aus der Datenbank
- Parameter:
  - country
  - limit
- Rückgabe:
  - Zeitstempel
  - Liste von Trends mit folgenden Attributen:
    * trendtype
    * name
    * country
    * placement
    * tweetVolume24

**vergangene Trends abrufen (GetRecentTrends)**
- Beschreibung: Lädt vergangene Trends aus der Datenbank
- Parameter:
  - hastag
  - country
  - startdate (optional)
  - enddate (optional)
- Rückgabe:
  - Liste an Trends mit folgenden Attributen:
    * Zeitstempel
    * Liste von Trends mit folgenden Attributen:
      · trendtype
      · name
      · country
      · placement
      · tweetVolume24

**Vorhandene Länder abrufen (GetAvailableCountries)**
- Beschreibung: Lädt vorhandene Länder aus der Datenbank
- Parameter: Keine
- Rückgabe: Liste mit allen Ländern, die in der Datenbak vorhanden sind

**Trends speichern (GetAvailableSentimentTrends)**
- Beschreibung: Lädt vorhandene Trends aus der Datenbank
- Parameter:
  - query
  - limit
  - country
- Rückgabe: Liste an vorhandenen Sentiments

**aktuelle Sentiment abrufen (GetCurrentSentiment)**
- Beschreibung: Lädt aktuellen Sentiment aus der Datenbank
- Parameter:
  - Trendname
  - Country
- Rückgabe: Sentiment

**vergangene Sentiment abrufen (GetRecentSentiment)**
- Beschreibung: Lädt vergangene Sentiment aus der Datenbank
- Parameter:
  - Trendname
  - Country
  - startdate (optional)
  - enddate (optional)
- Rückgabe: Liste an Sentiments

**DB auf Tweet prüfen (GetUniqueTweets)**
- Beschreibung: Gibt zurück, welche Tweets noch nicht in der Datenbank sind
- Parameter: Liste an Tweet IDs
- Rückgabe: Liste an Tweet IDs

Vom Backenend erzeugte Daten werden in einer MongoDB Datenbank gespeichert. Dabei werden Trends zu bestimmten Zeitpunkten gespeichert. Diese enthalten Trendtype, Trendname, Placement, Country, TweetVolume24. Für Sentiments werden die Tweets, die zu einem Trend gehören, und das jeweilige Topic gespeichert.

Um die Funktionalität zu gewährleisten wurden Unit Tests entworfen. Dadurch kann die interne Integrität, sowie die Anbindung an externe Systeme, sichergestellt werden. Die Tests wurden dabei mit dem Arrange-Act-Assert-Pattern entworfen. Dieses soll eine übersichtliche Struktur und Einheitlichkeit garantieren.

## VI. Frontend

## References


[1] Microservice-Frontend-Architekturen [Online] https://www.sigs-datacom.de/uploads/tx_dmjournals/attermeyer_OTS_Microservices_Docker_16.pdf (visited on Jun. 21, 2022)

[2] Twitter API [Online] https://developer.twitter.com/en/docs/twitter-api (visited on Jun. 23, 2022)

[3] TWINT - Twitter Intelligence Tool [Online] https://github.com/twintproject/twint (visited on Jun. 23, 2022)

[4] Tweepy: Twitter for Python! (2020) [Online] https://github.com/tweepy/tweepy (visited on Jun. 23, 2022)

[5] bert-base-multilingual-uncased-sentiment [Online] https://huggingface.co/nlptown/bert-base-multilingual-uncased-sentiment (visited on Jun. 23, 2022)

[6] TextBlob: Simplified Text Processing [Online] https://textblob.readthedocs.io/en/dev/ (visited on Jun. 23, 2022)