Reddiment: Reddit Sentiment-Analyse
Technical Report

Tobias Bauer
t.bauer@oth-aw.de

Fabian Beer
f.beer1@oth-aw.de

Daniel Holl
d.holl1@oth-aw.de

Ardian Imeraj
a.imeraj@oth-aw.de

Konrad Schweiger
k.schweiger@oth-aw.de

Philipp Stangl
p.stangl1@oth-aw.de

Wolfgang Weigl
w.weigl@oth-aw.de

***Zusammenfassung*—Dieser Technical Report beschreibt die Architektur von Reddiment – ein webbasiertes Dashboard zur Sentiment-Analyse von Subreddits.**

I. Einführung und Ziele

`r/wallstreetbets`, auch bekannt als WallStreetBets oder WSB, ist ein Subreddit, in dem über Aktien- und Optionshandel spekuliert wird. Der Subreddit ist bekannt für seine profane Art und die Vorwürfe, dass Nutzer/innen Wertpapiere manipulieren und volatile Kursbewegungen auslösen. Anhand von Sentiment-Analyse sollen nun Subreddits in Bezug auf Aktienkursverläufe analysiert werden. Dazu soll ein webbasiertes Dashboard entwickelt werden welches das Sentiment als auch die Erwähnungen bestimmter Schlüsselwörter in ausgewählten Subreddits über die Zeit und den Aktienverlauf gegenüberstellt.

In den weiteren Abschnitten des Technical Reports wird zuerst die Bausteinsicht des Gesamtsystems in Abschnitt II eingegangen. Im nächsten Abschnitt III wird die Verteilungssicht der Anwendung beschrieben. In Abschnitt IV werden die angewandten Werkzeuge zur Entwicklung der Anwendung vorgestellt. Abschließend wird kurz auf die angewandten Sicherheitskonzepte in Abschnitt V eingegangen und ein Fazit in Abschnitt VI gegeben, gefolgt vom Literaturverzeichnis am Ende.

II. Bausteinsicht

Diese Sicht zeigt die statische Zerlegung des Systems in Bausteine sowie deren Beziehungen.

A. Gesamtsystem

Reddiment bezieht Daten aus mehreren Quellen und stellt diese dem Benutzer aggregiert bereit. Die folgende Abbildung 1 zeigt die Interaktionen des Systems mit Fremdsystemen und dem Benutzer. Die beiden Datenquellen von Reddiment sind

- Reddit API für Subreddit Daten
- Yahoo Finance für Aktienmarkt Daten

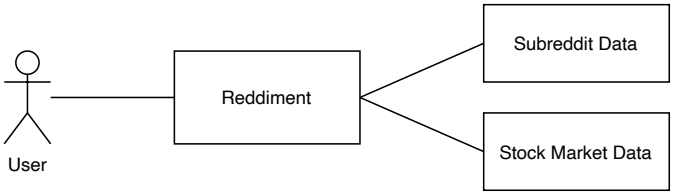

Abbildung 1. Kontextabgrenzung des Reddiment Gesamtsystems

B. Backend

Das Backend wurde als ein unter „Node.js“ [1] laufender Server realisiert. Hierfür kam das Web-Framework „Express“ [2] zum Einsatz. Darauf aufbauend wurde eine GraphQL-Schnittstelle [3] mithilfe des Frameworks „Apollo Server“ [4] implementiert. Die GraphQL-API ist unter der Route `/graphql` verwendbar.

Das Backend stellt die zentrale Kommunikations- und Verwaltungseinheit für die einzelnen Komponenten des Gesamtsystems Reddiment dar. So liefern die proaktiven Crawler (siehe Abschnitt II-D) die Rohdaten über eine GraphQL-Mutation an das Backend. Diese Rohdaten werden anschließend weiterverarbeitet und im Falle von Reddit-Kommentaren durch Aufruf des Sentiment-Services um den Sentiment-Wert erweitert. Alle so erhaltenen Daten werden über die Anbindung zur Datenbank (siehe Abschnitt II-C) persistiert.

Stellt nun das Frontend eine Anfrage an das Backend, werden diese aus der Datenbank geladen und aufbereitet. Die Gruppierung und Aggregation von beispielsweise Reddit-Kommentaren sowie die Interpretation des Sentiment-Werts erfolgt im Backend. Dies ermöglicht es uns, zukünftig eine feinere oder auch gröbere Zusammenfassung von Sentiment-Daten ohne Datenverlust oder aufwendigen Transformationen vorzunehmen.

Aufgrund des Einsatzes von GraphQL als Schnittstelle können Frontend und ggf. andere (externe) Akteure die benötigten Daten in einer von ihnen gewählten Struktur abfragen. Dies erlaubte eine flexible Entwicklung des Frontends ohne Kommunikations-Overhead. Weiterhin sind serverseitig die sogenannten Resolver modular aufgebaut, sodass verschiedene GraphQL-Felder in getrennten Quellcodedateien behandelt werden können. Alle Quellcodedateien im Zusammenhang mit GraphQL befinden sich im Ordner `backend/src/graphql`.

Das Backend ist aufgrund der zentralen Position im System Reddiment auf die anderen Bestandteile angewiesen.

Um dennoch Unit-Tests für ein isoliertes Backend schreiben und ausführen zu können, können die Datenbank und das Sentiment-Modul dynamisch durch Mocks ersetzt werden. Beim Start des Backends kann über die Umgebungsvariable `PRODUCTION` festgelegt werden, ob die realen Module (`true`) oder die Mock-Module (`false`) verwendet werden. Die entsprechenden Quellcodedateien liegen im Ordner `backend/src/services`. Da die Crawler proaktiv sind, müssen für diese keine Mocks erstellt werden – vielmehr simulieren die Unit-Tests die Crawler und prüfen, ob die gewünschten Daten von der GraphQL-Schnittstelle zurückgeliefert werden.

Im Ordner `backend/src/util` befinden sich verschiedene Quellcodedateien mit Hilfsfunktionen. Gestartet wird das Backend durch Ausführen der Datei `backend/server.ts`, die wiederum alle benötigten Module lädt und den Apollo Server mitsamt der GraphQL-Schnittstelle startet.

C. Datenbank

In der NoSQL Datenbank „Elasticsearch“ [5] werden Daten in Form von Dokumenten gespeichert. Für dieses Projekt werden persistent Daten, welche der Stock-Crawler und der Reddit-Crawler inklusive der Sentimentanalyse generiert, als Dokumente in der Datenbank gespeichert. Die Reddit-Daten bestehen aus den Inhalten: Subredditname, Kommentar, Zeitstempel, KommentarID, UserID, ArtikelID, Upvotes, Downvotes und Sentiment. Die Aktien-Daten setzten sich aus Aktienname, Zeitstempel, Eröffnungskurs, Maximum, Minimum, Schlusswert, dividendenadjustierte Zeitreihe und den Volumen zusammen. Um die Daten, welche in Dokumenten gespeichert werden, schneller in der Datenbank zu finden, arbeitet Elasticsearch mit Indizes. Dies bedeutet, dass ähnliche Dokumente unter dem gleichen Index gespeichert werden. Um eine Trennung der Dokumente herbeizuführen, werden Reddit-Daten mit dem Präfix f_ und Aktien-Daten mit dem Präfix f_ in der Datenbank gespeichert. Aus dieser Konvention ergibt sich für die Reddit-Daten der Index `r_subredditname` und für die Aktien-Daten `f_aktienname`. Um einen zeitlichen Verlauf einer Aktie oder des Sentiments zu erzeugen, müssen lediglich alle Dokumente eines gleichen Index abgerufen werden. Dieses Vorgehen ermöglicht es, dass bei einer Suchanfrage nicht die gesamte Datenbank durchsucht werden muss. Vielmehr genügt es, wenn Elasticsearch den Index prüft und alle Dokumente unter dem Zielindex ausgibt. Zusätzlich verwendet Elasticsearch eine ID für jedes Dokument, diese ist ein eindeutiger Indikator für das jeweilige Dokument. Bei Dokumenten, welche Reddit-Daten enthalten, wird die ID des Dokumentes gleich der eindeutigen KommentarID, welche vom Reddit-Crawler kommt, vergeben. Den Dokumenten mit Aktien-Daten wird folgende ID zugewiesen `Aktienname_Zeitstempel`. Mittels der Dokument ID können einzelne Dokumente eindeutig gesucht und ausgegeben werden.
Für die Kommunikation mit dem Backend wurde der *Node.js Client* für Elasticsearch implementiert. Der Client liefert eine persistente Verbindung zwischen der Datenbank und dem Backend.

D. Dienste

Dieser Abschnitt beschreibt die Dienste (engl. Services). Reddiment hat drei Dienste: Sentiment, Reddit Crawler und Stock Market Crawler.

1) Sentiment: In diesem Dienst wird für einen Text das Sentiment ermittelt. Um das Sentiment an das Backend zu übermitteln wurde eine Flask-REST-API entworfen. Diese weist einen Endpoint `sentiment` auf. An diesen Endpoint kann mittels eines Posts ein Text (JSON-Format) übertragen werden. Das Modul *sentiment* ermittelt die Stimmungslage des Textes und gibt das Ergebnis in JSON-Format zurück. Die Ermittelung des Sentiments erfolgt zweifach, durch zwei regelbasierte Verfahren. Das geschieht, um eine Absicherung der Richtigkeit des Wertes zu haben. Einerseits wird dafür *Vader* [6](Valence Aware Dictionary and Sentiment Reasoner) der Python-Bibliothek *nltk* verwendet, zum anderen *TextBlob* [7].

2) Reddit Crawler: Der Reddit Crawler verwendet „snoowrap“ [8], das eine JavaScript-Schnittstelle für den Zugriff auf jeden Reddit-API-Endpunkt bereitstellt. *snoowrap* unterliegt den API-Regeln der Reddit-API, worin u.a. das Rate-Limit auf 60 Anfragen pro Minuten bzw. 600 Anfragen alle 10 Minuten festgelegt ist. Um die Reddit-API nutzen zu können, muss eine Applikation über die URL *https://www.reddit.com/prefs/apps* angelegt und mit einem bestehenden Reddit-Konto verknüpft werden. Für die Applikation kann man zwischen drei verschiedenen Typen wählen: *web app*, *installed app* und *script*. Für den Reddit Crawler wurde eine *script*-Applikation angelegt und mit einem bestehenden Reddit-Konto verknüpft. Mit der Registrierung einer Applikation erhält man eine individuelle *Client-ID* sowie ein individuelles *Client-Secret*. Mit der ID und dem Secret, sowie den Anmeldedaten für das Reddit-Konto, kann eine Snoowrap-Instanz generiert werden, die als Requester für die Reddit-API fungiert.
Der Reddit-Crawler erhält in einem zeitlichen Intervall eine Liste von *Subreddits* vom Backend, die durchforstet und die *Comments* gesammelt werden sollen. Das Sammeln der Comment-Einträge erfolgt ebenfalls in zeitlichen Intervallen, um das Rate-Limit nicht zu überschreiten und einen Fehlerstatuscode auszulösen. Nach dem Sammeln werden die Daten im json-Format an das Backend gesendet. Die verwendeten Funktionen aus *snoowrap* sind unter https://not-an-aardvark.github.io/snoowrap/index.html zu finden.

3) Stock Market Crawler: Um den Verlauf des Sentiments in der Vergangenheit auch mit der tatsächlichen Marktlage vergleichen zu können wurde eine weitere REST-API für Marktdaten erstellt. Diese weißt den Endpoint `/post` auf und erwartet ein valides *Tickersymbol*(Aktienkürzel) einer Aktie. Der zugehörige Aktienkurs wird von *Yahoo-Finance* geladen und zurückgegeben.

E. Frontend

Dieser Abschnitt beschreibt die client-seitige Frontend-Architektur. Das Frontend wird unter Zuhilfenahme des Frontend-Frameworks SvelteKit [9] realisiert. Das Frontend besteht aus zwei Bausteinen: 1) den Routen zur Navigation, und 2) der *Library* für Komponenten und weitere Module.

Die Routen zur Navigation sind selbst in zwei Unter-Bausteine zerlegt und befinden sich im Ordner `routes`

1) Layout: Es gibt Elemente, die auf jeder Seite sichtbar sind, z. B. die Navigationsleiste oder eine Fußzeile. Anstatt sie auf jeder Seite zu wiederholen, wird eine Layout-Komponente namens `src/routes/__layout.svelte` verwendet.

2) Route: Die *Library* ist eine Sammlung von Frontend-Komponenten, ...

3) Components: In diesem Modul sind die Svelte-Komponenten gesammelt, mit denen die eigentliche Anzeige im Webbrowser realisiert wird. Die Komponenten behandeln alle Eingaben und kommunizieren bei Bedarf mit dem Backend über die GraphQL API Schnittstelle.

III. Verteilungssicht

Das Verteilungssicht beschreibt die Verteilung des Gesamtsystems, wichtige Begründungen für diese Verteilungsstruktur und die Zuordnung von Softwareartefakten zu Bestandteilen der Infrastruktur. Zentraler Bestandteile der Verteilungsstruktur sind „Docker“ [10] Container. Damit das gesamte System, mit allen verteilten Komponenten möglichst unkompliziert und in der korrekten Reihenfolge gestartet wird, werden die Docker Container durch „Docker-Compose“ verwaltet. Neben der unkomplizierten Bedienung finden zusätzlich zahlreiche weitere Vorteile Anwendung. Diese sind für u.a. die Vergabe von Netzwerkkonfigurationen wie Hostnames an die Anwendungen der jeweiligen Docker-Container oder aber auch für das Konfigurieren von globalen, einfach von dem Nutzer anzupassende Environment-Variablen. Sensible Daten wie Zugangspasswörter werden durch sogenannten „Docker-Secrets“ sicher zur Verfügung gestellt. Dies wird im Folgenden Abschnitt V-A genauer erläutert.

IV. Entwicklungswerkzeuge

Im folgenden Abschnitt wird auf die verwendeten Entwicklungswerkzeuge eingegangen.

A. Paketverwaltung

Die Verwaltung der Abhängigkeiten erfolgt mit „npm“ [11].

B. Linting

Im Frontend wird „eslint“ in Verbindung mit „prettier“ verwendet,um die Einhaltung der Codierrichtlinien zu gewährleisten. Die Konfigurationen sind jeweils in den Dateien `eslintrc.js` und `prettierrc.js` hinterlegt.

C. Build-Tools

1) Backend: Im Backend wird der Typescript Compiler „tsc“ verwendet, um die Dateien in ein Format zu überführen, welches mit `node` ausgeführt werden kann.

Die Konfiguration findet sich dabei in der Datei `tsconfig.json`.

2) Frontend: Im Frontend ist Vite dafür zuständig, die Anwendung aus dem Quellcode zu erstellen. Dabei gibt es zwei Varianten: Für Entwicklungszwecke wird ein Vite-Dev-Server (mit Reload-Funktionalität) zum Bereitstellen der Anwendung verwendet, während für den Produktiveinsatz nur die benötigten Zieldateien unter Verwendung des `Static adapter` erstellt werden, die dann mit einer beliebigen Server-Software ausgeliefert werden können.

Ferner sind Alias-Namen für häufig genutzte Verzeichnisse definiert, um Pfadangaben zu vereinfachen.

D. Unit Tests

Im Backend werden Unit-Tests anhand von „mocha“ [12] durchgeführt. Außerdem wird „nyc“ [13] für die Erzeugung der Test Abdeckung verwendet.

Im Frontend wird „Vitest“ [14] verwendet. Für die Frontend-Komponenten wird zusätzlich die „svelte-testing-library“ [15] verwendet. Diese ermöglicht es, die Komponenten zu *rendern* und Details über die verschiedenen Elemente innerhalb der Komponente zu erhalten.

V. Sicherheitskonzepte

Dieser Abschnitt beschreibt die angewendeten Sicherheitskonzepte, die für eine sichere Entwicklungs- und Produktionsumgebung notwendig sind.

A. Secrets Verwaltung

Die Anforderung sensible Daten jeder Anwendung individuell und zusätzlich einfach Konfigurierbar zur Verfügung zu stellen, lässt schnell auf den Einsatz von Environment-Variablen schließen. Dies hat jedoch einige sicherheitsrelevante Schwachstellen. Aus diesem Grund wurden die komponentenspezifischen Zugangsdaten mittels Docker-Secret zur Verfügung gestellt. Dabei werden im Gegenteil zu Environment-Variablen die Passwörter in einer Text-Datei gespeichert und in dem jeweiligen Docker gemountet bzw. zur Verfügung gestellt.

Ein weiteres zu beachtendes Kriterium in diesem Sicherheitskonzept ist der Zugriff auf die jeweiligen API-Schnittstellen der einzelnen Komponenten. Der Zugriff soll nur den berechtigten Anwendungen gestattet sein. Dafür wurde für jede Anwendung ein individueller „Access-Key“ erstellt. Der Zugriff wird durch Prüfen des Schlüssels bei jeder Abfrage gewährt bzw. unterbunden. Unberechtigte Dritte können dadurch die API-Schnittstelle nicht verwenden und ein Missbrauch wird dadurch unterbunden. Aufgrund zeitlicher Einschränkungen wurden die Access-Keys zwar zur Verfügung gestellt, aber die Implementierung noch nicht fertiggestellt.

VI. Fazit und Ausblick

Literatur

[1] node. “Node.js.” [Online]. (2022), Adresse: https://www.nodejs.org/.

[2] Express. “Express.” [Online]. (2022), Adresse: https://expressjs.com/.

[3] The GraphQL Foundation. “GraphQL – A query language for your API.” [Online]. (2022), Adresse: https://graphql.org/.

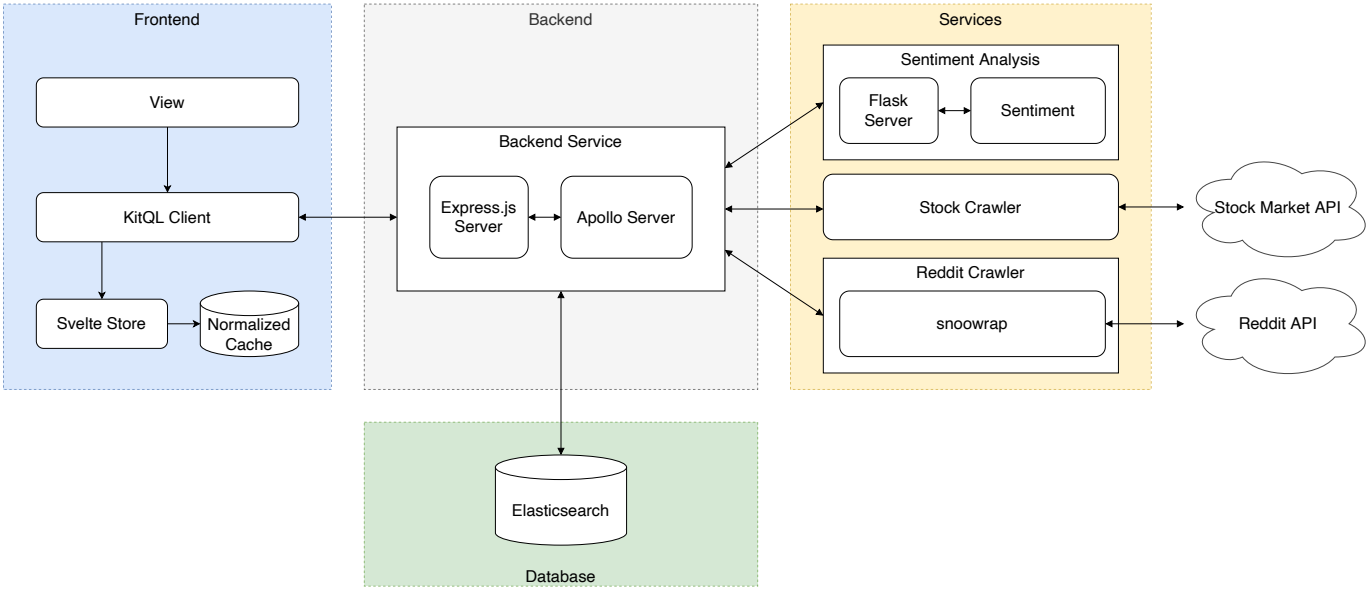

Abbildung 2. Überblick über die Architektur von Reddiment. Die Architektur besteht aus vier Teilen: Das Frontend bietet dem Nutzer ein graphisches Dashboard zur Visualisierung der Metriken (Abschnitt II-E), das Backend stellt (Abschnitt II-B) die GraphQL API bereit, die Datenbank (Abschnitt II-C) speichert persistent Reddit und Aktienmarkt Daten, und den Diensten (Abschnitt II-D) zur Sentiment Analyse als auch zum crawlen der Daten von APIs.

[4] Apollo Graph Inc. “Introduction to Apollo Server – Apollo GraphQL Docs.” [Online]. (2022), Adresse: https://www.apollographql.com/docs/apollo-server/.

[5] Elastic. “Elasticsearch.” [Online]. (2022), Adresse: https://www.elastic.co/de/elasticsearch/.

[6] C. Hutto und E. Gilbert, “Vader: A parsimonious rule-based model for sentiment analysis of social media text,” in *Proceedings of the international AAAI conference on web and social media*, Bd. 8, 2014, S. 216–225.

[7] textblob. “Sentiment Analysis.” [Online]. (2022), Adresse: https://textblob.readthedocs.io/en/dev/quickstart.html#sentiment-analysis.

[8] T. Katz. “snoowrap - A fully-featured JavaScript wrapper for the reddit API.” [Online]. (2022), Adresse: https://github.com/not-an-aardvark/snoowrap.

[9] Svelte. “Svelte Kit - The fastest way to build Svelte Apps.” [Online]. (2022), Adresse: https://kit.svelte.dev/.

[10] Docker. “Docker.” [Online]. (2022), Adresse: https://www.docker.com/.

[11] npm. “npm.” [Online]. (2022), Adresse: https://www.npmjs.com/.

[12] Mochajs. “Mocha - JavaScript test framework running on Node.js and in the browser.” [Online]. (2022), Adresse: https://mochajs.org/.

[13] Istanbul. “Istanbul - JavaScript test coverage made simple.” [Online]. (2022), Adresse: https://istanbul.js.org/.

[14] Vitest. “Vitest - A Vite-native unit test framework.” [Online]. (2022), Adresse: https://vitest.dev/.

[15] Testing Library. “Svelte Testing Library.” [Online]. (2022), Adresse: https://github.com/testing-library/svelte-testing-library.