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

Der Subreddit `r/wallstreetbets`, auch bekannt als WallStreetBets oder WSB, in dem über Aktien- und Optionshandel spekuliert wird. Er 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 den zeitlichen Verlauf von Sentiment und Erwähnungen bestimmter Schlüsselwörter in ausgewählten Subreddits dem 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 das Sicherheitskonzept in Abschnitt V vorgestellt und ein Fazit in Abschnitt VI gegeben.

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 für Reddiment sind

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

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.

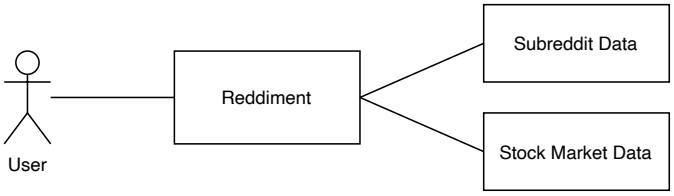

Abbildung 1. Kontextabgrenzung des Reddiment Gesamtsystems

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 Daten, welche an das Backend gesendet werden als Dokumente persistent in der Datenbank gespeichert. Die Reddit-Daten bestehen aus den Inhalten Die Aktien-Daten setzten sich aus Aktienname, Zeitstempel, Eröffnungskurs, Maximum, Minimum, Schlusswert, Dividenden-adjustierte Zeitreihe und den Volumen zusammen.

Um die Daten in der Datenbank zu organisieren, arbeitet Elasticsearch mit Indizes. Dies bedeutet, dass ähnlich strukturierte Dokumente unter demselben Index gespeichert werden. Pro Subreddit gibt es einen Index, dessen Name mit `r_` präfigiert ist, und pro Aktie gibt es einen Index mit Präfix `f_`. Um einen zeitlichen Verlauf einer Aktie oder des Sentiments zu erzeugen, müssen alle Dokumente eines Index abgerufen werden. Dieses Vorgehen ermöglicht es, dass bei einer Suchanfrage nicht alle Dokumente aller Indices durchsucht werden müssen, sondern eine Vorauswahl auf eine Untermenge getroffen werden kann.

Elasticsearch verwendet eine global eindeutige ID für jedes Dokument. Bei Dokumenten, welche Reddit-Daten enthalten, wird die ID des Dokumentes gleich der eindeutigen `KommentarID` gesetzt. Den Dokumenten mit Aktien-Daten wird folgende ID zugewiesen: `Aktienname_Zeitstempel`. Dokumente können auch anhand ihrer ID gesucht und ausgegeben werden. Dies wird beispielsweise bei der Aktualisierung von Dokumenten angewendet.

Im Backend wird das Paket „Elasticsearch Node.js client“ [6] verwendet. Das Paket enthält einen Client, der eine Verbindung zur Datenbank herstellt. In der Datei `backend/src/services/database.ts` befindet sich der Quellcode zur Kommunikation mit der Datenbank.

D. Dienste

Dieser Abschnitt beschreibt die Dienste (engl. Services) des Gesamtsystems Reddiment. Es gibt drei Dienste: Sentiment, Reddit-Crawler und Stock-Market-Crawler.

1) Sentiment: Dieser Dienst ermittelt für einen Text das Sentiment. Es wurde eine REST-API mit dem Endpunkt `/sentiment` implementiert. An diesen Endpunkt kann mittels einer POST-Anfrage ein Text übertragen werden. Der Dienst ermittelt anschließend die Stimmungslage des Textes und gibt das Ergebnis im JSON-Format zurück. Die Ermittelung des Sentiments erfolgt durch zwei regelbasierte Verfahren, um eine höhere Aussagekraft zu haben. Das erste regelbasiertes Verfahren ist „vader“ [7] (Valence Aware Dictionary and Sentiment Reasoner) der Python-Bibliothek *nltk*. Das zweite Verfahren ist „TextBlob“ [8]. Sind sich die beiden Verfahren in der Auswertung einig, wird der Sentiment-Wert von vader zurückgegeben. Andernfalls wird 0 (neutral) zurückgegeben.

2) Reddit-Crawler: Der Reddit-Crawler verwendet „snoowrap“ [9], das JavaScript-Funktionen für den Zugriff auf die Reddit-API bereitstellt. „snoowrap“ unterliegt den API-Regeln von Reddit, worin u.a. das Rate-Limit auf 60 Anfragen pro Minute. Um die Reddit-API nutzen zu können, muss ein API-Key über das Benutzerkonto [1] angefordert werden. Ein API-Key besteht aus den zwei Teilen: einer *Client-ID* und einem *Client-Secret*.

Mit dem API-Key und den Anmeldedaten für das zugehörige Reddit-Konto, kann ein Objekt der Klasse `Snoowrap` erstellt werden.Mit diesem Objekt können Reddit-API-Aufrufe durchgeführt werden.

Der Reddit-Crawler frägt in einem zeitlichen Intervall beim Backend an und erhält eine Liste mit Subreddit-Namen. Kommentare dieser Subreddits sollen von Reddit abgefragt und an das Backend gesendet werden. Das Sammeln der Kommentare erfolgt zyklisch, um das Rate-Limit nicht zu überschreiten. Pro Sammelzyklus werden die Kommentare an das Backend übermittelt.

3) Stock-Market-Crawler: Um den Verlauf des Sentiments in der Vergangenheit auch mit der tatsächlichen Marktlage vergleichen zu können, gibt es einen reaktiven Stock-Market-Crawler mit einer REST-API für Marktdaten. Dieser hat den Endpunkt `/post` und erwartet ein valides *Tickersymbol*(Aktienkürzel) einer Aktie. Der zugehörige Aktienkurs wird von *Yahoo-Finance* abgefragt und zurückgegeben.

E. Frontend

Dieser Abschnitt beschreibt die client-seitige Frontend-Architektur. Das Frontend wird unter Zuhilfenahme des Frontend-Frameworks SvelteKit [10] 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, ...

[1] https://www.reddit.com/prefs/apps

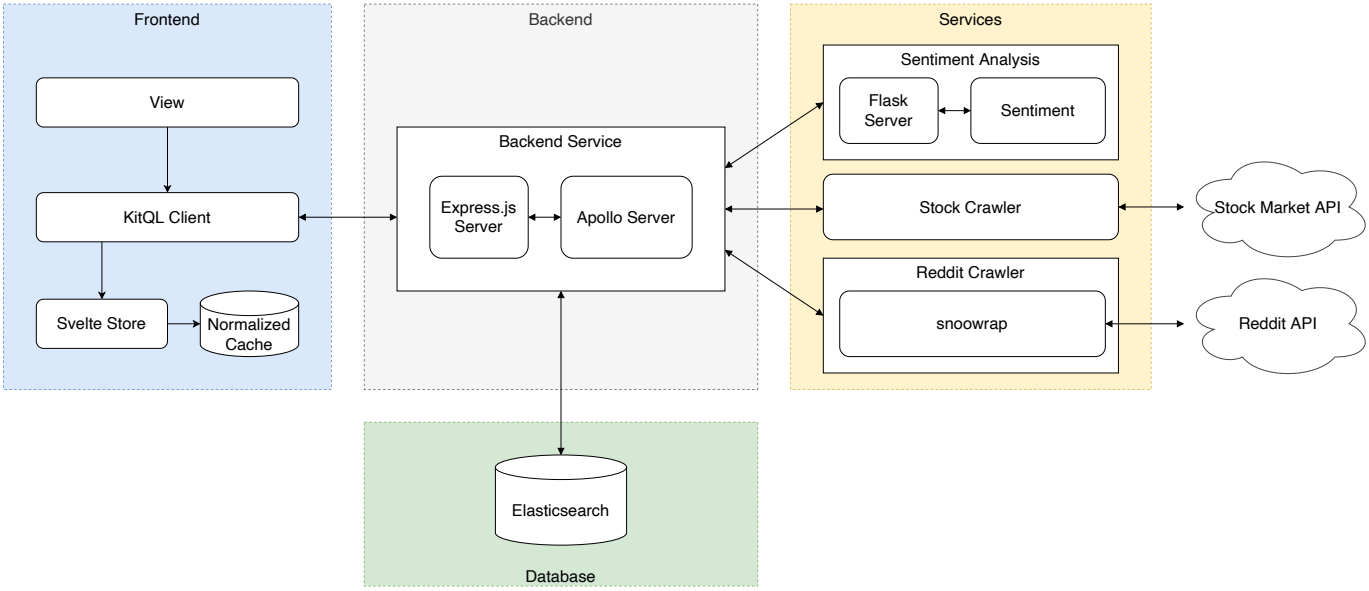

Abbildung 2. Überblick über die Architektur von Reddiment. Die Architektur besteht aus vier Teilen: Das Frontend bietet dem Nutzer ein graphisches Dashboard zur Visualisierung der Metriken (Abschnitt II-E), das Backend stellt (Abschnitt II-B) die GraphQL API bereit, die Datenbank (Abschnitt II-C) speichert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

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

Zentraler Bestandteil der Verteilungsstruktur sind Docker-Container [11]. Jeder Baustein von Reddiment ist für sich isoliert in einem eigenen Docker-Container untergebracht. Damit das gesamte System mit allen verteilten Komponenten in der korrekten Reihenfolge gestartet wird, werden die Docker-Container mit Docker Compose orchestriert. Docker Compose übernimmt die Netzwerkkonfiguration und die Vergabe von Host-Namen an die jeweiligen Docker-Container. Darüber hinaus können Umgebungsvariablen verwaltet werden. Sensible Daten, wie Zugangspasswörter, werden durch spezielle Mechanismen (siehe Abschnitt V) sicher zur Verfügung gestellt.

IV. Entwicklungswerkzeuge

In diesem Abschnitt wird auf die verwendeten Entwicklungswerkzeuge genauer eingegangen.

A. Paketverwaltung

Die Verwaltung der Abhängigkeiten erfolgt mit „npm“ [12] für auf „Node.js“ basierende Bausteine. Für den Sentiment-Baustein wird „Pipenv“ [13] verwendet.

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: Sowohl das Backend als auch die Crawler werden mit dem TypeScript Compiler „tsc“ [14] in ein für „Node.js“ ausführbares Format überführt. Die Konfiguration befindet 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.

D. Unit-Tests

Im Backend werden Unit-Tests mit „mocha“ [15] ausgeführt. Dabei wird „istanbul“ **istanbuljs** für die Erzeugung der Test-Abdeckung verwendet.

Im Frontend wird „Vitest“ [16] verwendet. Für die Frontend-Komponenten wird zusätzlich die „svelte-testing-library“ [17] eingesetzt. Diese ermöglicht es, die Komponenten zu *rendern*, um Details über die verschiedenen Elemente innerhalb der Komponente zu erhalten.

V. Secrets-Verwaltung

Die Anforderung, sensible Daten jeder Anwendung individuell und zusätzlich einfach konfigurierbar zur Verfügung zu stellen, lässt schnell auf den Einsatz von Umgebungsvariablen schließen. Ein solches Vorgehen birgt jedoch einige sicherheitsrelevante Schwachstellen. Aus diesem Grund werden die komponentenspezifischen Zugangsdaten

mittels dem Schlüsselwort `secrets` in der Docker-Compose-Konfigurationsdatei den Docker-Containern zur Verfügung gestellt. Dabei werden im Gegensatz zu Umgebungsvariablen die Passwörter in einer Text-Datei gespeichert und in dem jeweiligen Docker-Container eingebunden.

Ein weiteres zu beachtendes Kriterium in diesem Sicherheitskonzept ist der Zugriff auf die jeweiligen API der einzelnen Komponenten. Beispielsweise sollen nur die Crawler die GraphQL-Mutations des Backends verwenden dürfen. Dafür wurde für jede Anwendung ein individueller „Access-Key“ erstellt. Der Zugriff wird durch Prüfen des Schlüssels bei jeder Anfrage gewährt bzw. unterbunden. Unberechtigte Dritte können dadurch die API nicht verwenden und ein Missbrauch wird dadurch erschwert. Aufgrund zeitlicher Einschränkungen konnte das Sicherheitskonzept nicht vollständig implementiert werden.

VI. Fazit und Ausblick

Literatur

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

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

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

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

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

[6] ——, (2022). “Elasticsearch Node.js client.” [Online], Adresse: https://www.npmjs.com/package/@elastic/elasticsearch.

[7] 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.

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

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

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

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

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

[13] Python Packaging Authority. (2022). “Pipenv: Python Dev Workflow for Humans.” [Online], Adresse: https://pipenv.pypa.io/en/latest/.

[14] Microsoft. (2022). “TypeScript: JavaScript With Syntax For Types.” [Online], Adresse: https://www.typescriptlang.org/.

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

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

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