Fachhochschule Südwestfalen
University of Applied Sciences

Entwicklung eines Telegram-Bots zur Abfrage von Informationen aus der Software Graylog per Sprachnachricht

Development of a Telegram Bot to Retrieve Information From Graylog Software via Voice Message

Bachelorarbeit

Jonas Michel Berger

Fachhochschule Südwestfalen

2. November 2022

Autor: Jonas Michel Berger
Referent: Prof. Dr. Hans-Georg Eßer
Korreferent: Prof. Dr. Heiner Giefers
Eingereicht: 2. November 2022

Zusammenfassung

Das Thema dieser Abschlussarbeit ist die Entwicklung eines virtuellen Assistenten (Bot) für den Telegram Messenger, welcher Netzwerkadministratoren bei ihrer täglichen Arbeit durch die Möglichkeit unterstützen soll, in Echtzeit Informationen zu ihrem verwalteten Netzwerk abzurufen. Dabei wird auf die Software Graylog zurückgegriffen, welche die zentrale Verwaltung von anfallenden Systemprotokollen für ein Netzwerk übernimmt. Der Systemadministrator kann vordefinierte Abfragen in der Bot-Software hinterlegen und verknüpfen, sodass diese auf Anfrage jederzeit an die in Graylog integrierte Suchmaschine übermittelt werden können. Die Kommunikation zwischen Bot und Anwender erfolgt über Sprachnachrichten. Um die Nachrichten verarbeiten zu können, wandelt der Bot eingehende Sprachnachrichten mit einer Spracherkennung in Text um.

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Werken anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

2. November 2022

Jonas Michel Berger

Inhaltsverzeichnis

1. Einleitung

Unternehmen haben heutzutage einen großen Anspruch an eine IT-Umgebung, welche möglichst fehler- und unterbrechungsfrei funktioniert. Um den Status der Geräte und Systeme in einem Firmennetzwerk bestmöglich überwachen und somit Fehler frühzeitig erkennen zu können, stehen den Systemadministratoren zahlreiche Werkzeuge zur Verfügung, welche die Überwachung auf Fehlerzustände eigenständig übernehmen und den Verantwortlichen somit eine mühselige und monotone Arbeit ersparen. Für die Überwachung der Systeme werden häufig zwei verschiedene Systeme verwendet.

Ein klassisches Monitoring-System überwacht im festgelegten Sekunden- oder Minutentakt einzelne Systeme auf ihre Erreichbarkeit und Funktionsfähigkeit. Weiterhin können im Monitoring Zustände wie der Füllungsgrad der Datenspeicher oder die Auslastung von CPU- und Arbeitsspeicherressourcen überwacht werden. Das Monitoring-System sorgt so für eine Überwachung der Betriebszustände der Systeme rund um die Uhr.

In vielen Netzwerken wird zusätzlich ein Logserver eingesetzt, welcher die anfallenden Systemprotokolle der Geräte im Netzwerk zentral aufnimmt und durchsuchbar macht. Jede Software besitzt eine Art von Fehlerausgabe, welche je nach Einsatzzweck in das zentrale Systemprotokoll des Betriebssystems oder eine Textdatei geschrieben wird. Im Zeitalter von virtualisierten Umgebungen und Microservices hat sich die Anzahl der eigenständigen Systeme in einem Firmennetzwerk vervielfacht. Häufig werden Dienste in die Cloud ausgelagert, da dort Ressourcen besser skaliert werden können und dabei keine einmaligen und fortlaufenden Kosten für die Anschaffung und Wartung neuer Hardware entstehen. Bei der Fehleranalyse eines Dienstausfalls können die Protokolle vom Logserver bezogen werden. Erzeugt ein System aufgrund des vorhergehenden Ausfalls eines anderen Systems eine Fehlermeldung, ist dieser zeitliche Zusammenhang in der kumulierten Ansicht über den Logserver deutlich nachvollziehbar.

Das Produkt Graylog Open[1] stellt Administratoren eine quelloffene und kostenlose Lösung zur Verfügung, welche eine Vielzahl von Schnittstellen für die Anbindung an Systeme in einem Netzwerk bietet. Für die Bedienung von Graylog und den Zugriff auf die erfassten Protokolle steht eine HTTP-Webschnittstelle für den Webbrowser zur Verfügung. Weiterhin bietet das Produkt eine REST-API für den programmgesteuerten Zugriff an. Mit dieser Schnittstelle können vielseitige benutzerdefinierte Lösungen entwickelt werden, wie eine Aufbereitung der Daten mittels Diagrammen oder der Anschluss der Software an ein Selfservice Portal. In Graylog können Regeln zur Aufbereitung der erfassten Daten definiert werden. Reguläre Ausdrücke ermöglichen es, bestimmte Parameter aus Meldungen zu extrahieren und die Meldungen so zu kategorisieren. Später können Suchen zu bestimmten Kategorien über alle erfassten Meldungen durchgeführt werden.

[1] `https://www.graylog.org/products/open-source`

1.1. Ziel der Bachelorarbeit

Es soll die Entwicklung eines Bots für den Telegram-Messenger in der Programmiersprache Python geplant und durchgeführt werden. Der Bot interagiert mit dem Benutzer über gesprochene Sprache, indem der Anwender als Administrator eines Netzwerks die gewünschten Informationen zu Systemen im Netzwerk in einer Sprachnachricht beschreibt und der Bot mit einer per Sprachsynthese erstellten Nachricht antwortet. Die Informationsquelle ist eine Installation des Logservers Graylog Open. Es wird vorausgesetzt, dass die Software bereits für den Betrieb im Firmennetzwerk eingerichtet wurde und sämtliche zu überwachende Systeme angeschlossen wurden. Der Anwender kann beispielsweise Informationen zu Webservern im Firmennetzwerk anfordern: „Liefere mir Informationen zu Fehlern bei Webservern im Zeitraum der letzten 24 Stunden“. Der Bot soll nun die Inhalte der Nachricht in eine Anfrage an die in Graylog integrierte Suchmaschine umwandeln und dem Anwender das Ergebnis aufbereitet wiedergeben, z.B.: „Es wurden 30 Ereignisse gefunden“.

Zu den Aufgaben der zu entwickelnden Software gehört die Koordination der Ereignisse inklusive der Behandlung von Fehlerfällen sowie die Kommunikation mit dem Benutzer.

1.2. Aufbau der Bachelorarbeit

In Kapitel 2 werden die technischen Grundlagen erläutert. Hierzu zählen insbesondere die verwendeten Produkte und eine technische Erläuterung, wie die Systeme in der zu entwickelnden Software zusammenarbeiten. In Kapitel 3 wird der Planungsprozess erläutert. Dabei wird auf Modelle des Softwareengineering zurückgegriffen sowie der Programmablauf mit einem UML-Diagramm erläutert. Schließlich werden Entscheidungen erläutert, welche bei der Planung getroffen wurden. In Kapitel 4 wird die Implementierung behandelt. Das Kapitel beschreibt zunächst die Einrichtung der Softwareentwicklungsumgebung und die Implementierung erster funktionaler Prototypen. Außerdem wird ein Konzept erläutert, welches zum Zeitpunkt der Umsetzung der Theorie in die Praxis erstellt wurde. In Kapitel 5 wird der praktische Einsatz der Software beschrieben und einen Aspekt der IT-Sicherheit erläutert. In Kapitel 6 erfolgt eine Zusammenfassung und ein Ausblick auf mögliche technische Optimierungen.

2. Grundlagen

In diesem Kapitel werden Technologien und Produkte erläutert, auf welchen die zu implementierende Software basiert.

2.1. REST

REST ist eine Abkürzung für ‚Representational State Transfer' [1, S. 76 ff.]. Es handelt sich hierbei um ein Vorbild für einen Softwarearchitekturstil, welches in der Entwicklung verteilter Systeme Anwendung findet. REST-APIs werden für viele Dienste des Internets meist unsichtbar für die Anwender für die Kommunikation zwischen informationstechnischen Systemen verwendet. In der englischen Sprache wird häufig das Adjektiv ‚RESTful' für Ressourcen verwendet, welche die REST-Prinzipien anwenden [2, S. 277].

2.1.1. Anwendung in der Praxis

Heutzutage stellen viele Dienste HTTP-APIs zur Verfügung, welche als ‚RESTful' bezeichnet werden können. Die Kommunikation mit den Programmierschnittstellen beschränkt sich dabei auf die HTTP- und HTTPS-Protokolle, welche auch für den Abruf von Webinhalten im Browser verwendet werden. Durch die hohe Verbreitung wird die Implementierung neuer vernetzter Applikationen für Softwareentwickler stark vereinfacht.

Der Zugriff auf HTTP-APIs erfolgt analog zum Zugriff auf Webseiten. Je nach Anforderung werden die HTTP-Methoden ‚GET', ‚POST' und weitere verwendet [2, S. 279]. Die von der HTTP-API beziehbaren Informationen werden mit der URL (Uniform Resource Locator) lokalisiert und angefordert. Zusätzlich können Informationen über Header (Kopfzeilen) und Cookies ausgetauscht werden, um beispielsweise Zustandsinformationen (Status des Warenkorbs in einem Webshop, Identifikationsnummer des angemeldeten Benutzers) übertragen zu können.

Listing 2.1: Beispiel eines Aufrufs der Graylog REST-API.

```
GET http://10.10.12.1:9000/api/cluster
```

```
{
  "12ac8e44-0c59-4c38-bd5e-4fe247a55893": {
    "facility": "graylog-server",
    "codename": "Noir",
    "node_id": "12ac8e44-0c59-4c38-bd5e-4fe247a55893",
    "cluster_id": "8421ad08-fe62-4a04-8576-a81cab1a9c55",
    "version": "4.3.6+36120cf",
    "started_at": "2022-09-10T23:15:05.435Z",
    "hostname": "graylog.home.arpa",
```

```
10      "lifecycle": "running",
11      "lb_status": "alive",
12      "timezone": "Europe/Berlin",
13      "operating_system": "Linux 5.4.0-125-generic",
14      "is_processing": true
15    }
16  }
```

Die Antwort der HTTP-API erfolgt in vielen Fällen maschinenlesbar in der Sprache JSON [2, S. 281], wie in Listing 2.1 ab der ersten Zeile zu sehen.

2.2. Telegram Messenger

Der Dienst Telegram bietet seinen Nutzern eine Möglichkeit um kostenlos miteinander u. a. über Apps für die Mobilbetriebssysteme Android und iOS zu kommunizieren. Telegram hatte nach nach eigenen Angaben weltweit im Juni 2022 erstmals über 700 Millionen monatliche Nutzer und befand sich unter den fünf Apps mit den höchsten Downloadzahlen[1]. Der Dienst stellt damit eine beliebte Alternative zu anderen Messengern wie WhatsApp oder Signal dar. Ein Alleinstellungsmerkmal von Telegram gegenüber WhatsApp oder Signal sind die Bots und die dazugehörige API.

2.2.1. Bots

Der Begriff Bot stammt von dem englischen Begriff ‚robot' ab und bezeichnet ein Programm, welches einfache und wiederkehrende Aufgaben selbstständig erledigt. Bots werden heutzutage an vielen Stellen eingesetzt, beispielsweise in Telefonwarteschleifen oder in Webshops in Form von virtuellen Beratern für den Verkauf oder den Service. Der Telegram Messenger bietet Softwareentwicklern die Möglichkeit, mittels einer HTTP-API einen Bot auf vielfältige Weise an eigene Software anzubinden. Der Bot agiert Benutzern auf der Telegram-Plattform gegenüber wie ein normaler Verbindungspartner mit dem Unterschied, dass dieser zuerst mit einer Initialnachricht (‚/start') vom Benutzer aktiviert werden muss. So wird verhindert, dass Benutzer ohne Zustimmung kontaktiert werden können. Bots können auch zu Gruppen hinzugefügt werden. Dem Entwickler stehen zahlreiche und ausführlich dokumentierte API-Funktionen[2] zur Verfügung, sodass die vom Bot ausgehende Kommunikation nicht auf Textnachrichten limitiert ist. Bots stehen u. a. folgende Möglichkeiten zur Verfügung:

- Senden und Empfangen von Audio-, Bild- und Videodateien, Sprachnachrichten und Standortdaten
- Erstellen von Umfragen in einer Gruppe
- Abfrage von Informationen der aktiven Kommunikationspartner: Profilbild, Name, Zeitpunkt der letzten Erreichbarkeit (sofern vom Benutzer freigegeben)

[1] `https://telegram.org/blog/700-million-and-premium`
[2] `https://core.telegram.org/bots/api#available-methods`

- Gruppenadministration: Setzen und Entfernen von MOTD-Nachrichten (dt. „Mitteilungen des Tages“)
- Vereinfachte Bedienung durch vom Entwickler definierte Antwortmöglichkeiten, welche durch den Anwender direkt mittels Buttons gewählt werden können

Aufgrund der weitreichenden Möglichkeiten, der kostenlosen Benutzbarkeit und der für Anwendungsfälle mit bis zu 30 API-Zugriffen pro Sekunde ausreichenden Durchsatzratenbegrenzung[3] steht eine Vielzahl von Bots für die Benutzer der Plattform zur Verfügung. Als Beispiele seien genannt der Bot „WDR aktuell“ <`@WDRaktuell_bot`> für den Abruf von regionalen Informationen in Nordrhein-Westfalen, der von Google angebotene und daher als „verifiziert“[4] markierte „Gmail Bot“ <`@GmailBot`> für den Nachrichtenabruf und das Senden von Antworten für Gmail sowie der Bot IFTTT („if this then that“) <`@IFTTT`>, welcher die Automatisierung und Verknüpfungen von Webanwendungen erleichtert.

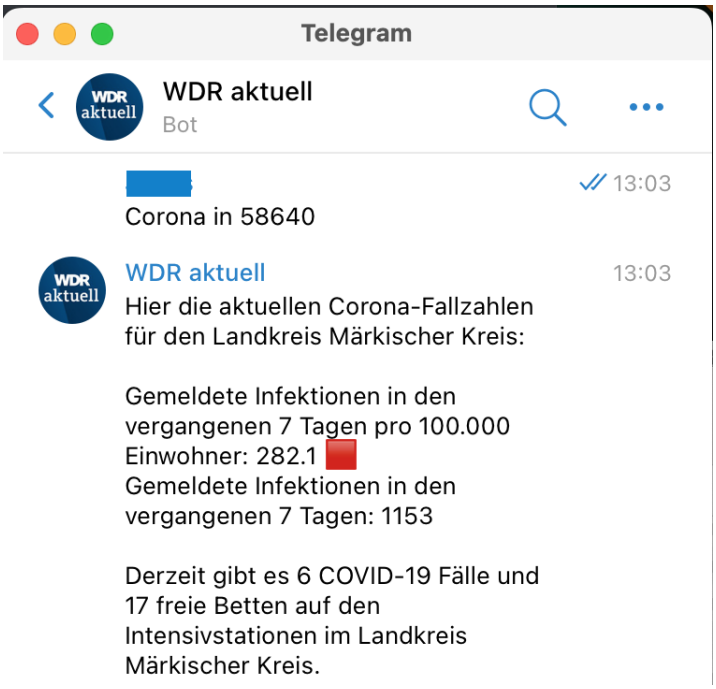

Abbildung 2.1.: Interaktion mit dem Bot von WDR aktuell in Telegram für macOS.

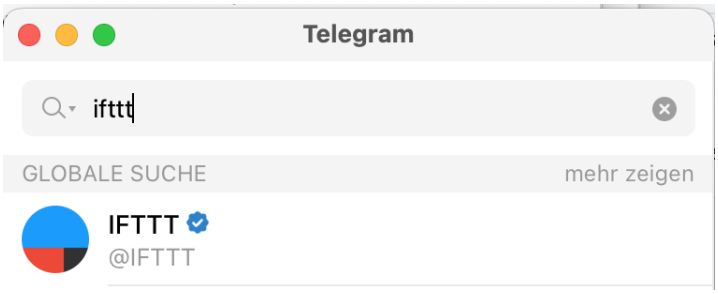

Abbildung 2.2.: Verifizierter IFTTT-Bot in der Suche von Telegram für macOS.

[3] `https://core.telegram.org/bots/faq#my-bot-is-hitting-limits-how-do-i-avoid-this`
[4] `https://telegram.org/verify`

Beziehen von Aktualisierungen von der Telegram API

Laut der technischen Dokumentation der Telegram API[5] stehen zwei Möglichkeiten zur Verfügung, um Aktualisierungen zu erhalten: Long-polling mit der API-Methode `getUpdates()` oder die Verwendung eines Webhooks. Die beiden Möglichkeiten verwenden unterschiedliche Konzepte und bieten damit verschiedene Vor- und Nachteile.

Im Fachbereich der Informatik steht der Begriff Polling für eine dauerhafte wiederkehrende Abfrage von Informationen bei einem Dienst. Diese Technik beansprucht viel CPU-Zeit eines Systems und wird aufgrund des geringen Gegenwerts der meist leer ausgehenden Abfragen als teuer bezeichnet. Long-polling beschreibt eine Technik, bei welcher der Client einer HTTP-Abfrage einen verlängerten Zeitraum bis zum Erhalt der Antwort vom Server akzeptiert, ohne die Abfrage aufgrund einer Zeitüberschreitung abzubrechen. Die Länge des Zeitraums ist variabel. Long-polling bietet damit gegenüber dem klassischen Polling den Vorteil, dass die benötigte Rechenzeit durch die verlängerten Abfragezeiträume stark reduziert wird.

Die Software- und Betriebssystementwicklung bietet eine Technik, um die Verwendung von teurem Polling zu umgehen: die Verwendung von Interrupts oder Callbacks (dt. Rückruf). Hierbei erhält der Absender der Anfrage eine Rückmeldung, sobald die Antwort vorliegt. Der Vorteil dieser Technik ist, dass nach dem Absenden der Anfrage keine CPU-Zeit seitens des Absenders notwendig ist. Es muss jedoch eine Möglichkeit vorhanden sein, den Absender im Falle einer eingehenden Antwort zum Vorgang zurückzuführen, sodass die weitere Bearbeitung möglich wird. Im Falle der Anbindung der Telegram API übernimmt dies der Webhook. Der Webhook muss (durch die Telegram-API) öffentlich aus dem Internet erreichbar und durch den Verwalter des Bots im Vorhinein durch eine entsprechende URL definiert worden sein. Wird eine Nachricht an den Bot gesendet, prüft die Telegram-API intern, ob Definitionen für Webhooks vorliegen. Im positiven Fall sendet die Telegram API eine HTTP-Anfrage an die URL des Webhooks mit den Details zur eingegangenen Nachricht. Nach Erhalt der Anfrage durch den Webhook muss dieser die Daten zur verarbeitenden Software zurückführen und die weitere Bearbeitung auslösen.

Erstellung eines Bots

Um ein Programm über die Telegram Bot-API anbinden zu können, muss zuvor ein Bot über Telegram erstellt werden. Hierzu wird der Bot `@BotFather` verwendet. Sämtliche Einstellungen zu Bots werden über `@BotFather` vorgenommen. Soll ein neuer Bot erstellt werden, werden die benötigten Informationen abgefragt. Nachdem der Anzeigename und der Benutzername festgelegt werden, erhält der Benutzer die Zugangsdaten für die HTTP-API und der Bot ist einsatzbereit. `@BotFather` kann jederzeit erneut kontaktiert werden, um weitere Einstellungen vorzunehmen: es können Profilbild und Beschreibung geändert, sowie vordefinierte Kommandos festgelegt werden. Der Zugriff auf die HTTP-API erfolgt über die Basis-URL `https://api.telegram.org/bot<token>/METHOD_NAME`[6], wobei `<token>` der von `@BotFather` erhaltene Zugriffsschlüssel und `METHOD_NAME` die API-Methode (beispielsweise `getUpdates` oder `sendMessage`) ist. Weitere laut API-Dokumentation benötigte Parameter werden als Parameter der HTTP-Anfragen vom Typ ‚GET' oder ‚POST' übermittelt. Die Antwort der API erfolgt in Form eines JSON-Objekts.

[5] `https://core.telegram.org/bots/api#getting-updates`
[6] `https://core.telegram.org/bots/api#making-requests`

Listing 2.2: Beispiel eines Aufrufs der Telegram HTTP-API. Erhalt einer Textnachricht „Hallo Welt“.

```
GET https://api.telegram.org/bot123456:ABC-DEF1234ghIkl-
    zyx57W2v1u123ew11/getUpdates
```

```
{
    "ok": true,
    "result": [
        {
            "update_id": 987654321,
            "message": {
                "message_id": 626,
                "from": {
                    "id": 12345678,
                    "is_bot": false,
                    "first_name": "Jonas",
                    "username": "**entfernt**",
                    "language_code": "de"
                },
                "chat": {
                    "id": 12345678,
                    "first_name": "Jonas",
                    "username": "**entfernt**",
                    "type": "private"
                },
                "date": 1662808990,
                "text": "Hallo Welt"
            }
        }
    ]
}
```

2.3. Graylog Open

Graylog Open ist ein Softwareprodukt der Firma Graylog, Inc mit dem Hauptsitz in Houston, Texas in den USA. Das Produkt stellt eine kompakte Verwaltungsoberfläche für die Erfassung von Systemprotokollen bereit, welche in einer Elasticsearch Suchmaschine vorgehalten werden. Der Quellcode der Software kann öffentlich eingesehen werden[7].

Hohe Anforderungen an die (Ausfall-)sicherheit moderner und komplexer IT-Systeme führen zu der Notwendigkeit, die Funktionsfähigkeit der Systeme möglichst allumfassend und automatisiert zu prüfen. Herkömmliche Monitoringsysteme mit einem Fokus auf die Erreichbarkeit oder die Überwachung von vordefinierten Fehlerausgaben erfüllen diese Anforderungen nicht. Fehlerzustände sollen in Echtzeit, möglichst vor und spätestens zum Zeitpunkt einer durch den Anwender spürbaren Einschränkung auffallen. Informationstechnische Systeme in sämtlichen Bereichen sind längst zu komplex geworden, um alle Fehlerquellen im Vorhinein bestimmen

[7] https://github.com/Graylog2/graylog2-server

und gezielt überwachen zu können. Aus diesem Grund wird ein anderer Ansatz als beim herkömmlichen Monitoring angewendet: die Erfassung der Systemprotokolle der zu überwachenden Systeme. Diese ermöglichen einen umfassenderen Blick auf die aktuellen Ereignisse. Durch die Anwendungsprotokolle können Fehlerzustände eines Webservers beispielsweise bereits nach dem ersten Besuch eines Besuchers einer durch den fehlerhaften Webserver beeinträchtigten Webseite erkannt werden.

2.3.1. Erfassung von Systemprotokollen

Graylog Open ist kompatibel zu einer Vielzahl von Betriebssystemen und Anwendungen. Linuxbasierte Systeme verwenden häufig einen Dienst zur zentralen und systemweiten Erfassung der Anwendungsprotokolle, welcher das in RFC 3164 standardisierte Protokoll Syslog[8] verwendet. Dieser schreibt in der Standardkonfiguration vieler aktueller Betriebssysteme auf Basis der Linux-Distribution Debian alle Meldungen in Textform in die Datei `/var/log/syslog`, bei Systemen auf Basis der Linux-Distribution Red Hat Enterprise Linux in die Datei `/var/log/messages`. Die Position dieser Ausgabedatei auf dem Dateisystem ist für die Verarbeitung der Meldungen in Graylog jedoch nicht wichtig, da die Meldungen nach einer Anpassung der Konfiguration des Dienstes (offiziell wird nur die Software Rsyslog und Syslog-ng von Graylog unterstützt[9]) über das Netzwerk per UDP und mit optionaler TLS-Verschlüsselung an Graylog übertragen werden. In Graylog muss hierzu die Annahme von Daten über das Netzwerk mit dem syslog-Protokoll aktiviert werden. Mittels einer Input-Konfiguration wird ein Port an der zum überwachenden System nächstgelegenen Netzwerkschnittstelle eröffnet, auf welchem die Software auf eintreffende Meldungen im syslog-Format lauscht.

Auch mit Docker bereitgestellte Microservices können global überwacht werden, ohne die Konfiguration der Container oder sogar die der Anwendungen in einem gestarteten Container einzeln anpassen zu müssen. Hierzu wird nicht das syslog-Protokoll, sondern das von Docker ohnehin unterstützte[10] Protokoll GELF verwendet. Es ist eine zentrale Änderung der Konfiguration des Docker-Dienstes notwendig, um fortan die Protokolle aller neu gestarteten Container über das wahlweise UDP- oder TCP-basierte GELF-Protokoll an die Graylog-Instanz zu senden. Die an Graylog übertragenen Daten entsprechen der Ausgabe des Kommandos `docker logs`. Um den Empfang von Daten über das GELF-Protokoll seitens Graylog zu ermöglichen, ist es analog zur Einrichtung für das syslog-Protokoll notwendig, mittels einer Input-Konfiguration einen Port auf einer Netzwerkschnittstelle zu reservieren.

Systemprotokolle aus Windows können nicht direkt verarbeitet werden. Es ist der Einsatz einer Middleware wie Winlogbeat notwendig, welche die Protokolle auf dem System erfasst und die Daten über ein Netzwerkprotokoll an Graylog sendet.[11]

[8] `https://www.ietf.org/rfc/rfc3164.txt`
[9] `https://docs.graylog.org/docs/syslog`
[10] `https://docs.docker.com/config/containers/logging/gelf/`
[11] `https://docs.graylog.org/docs/windows`

2.3.2. Verarbeitung

Für die Verarbeitung der erfassten Daten stellt Graylog dem Systemadministrator eine Vielzahl von Möglichkeiten zur Verfügung. Hier werden lediglich die Möglichkeiten erläutert, welche für das Ziel der Abschlussarbeit, also der sprachgesteuerten Verarbeitung durch den Telegram-Bot, relevant sind. Graylog erhält die Daten verschiedener Systeme über die konfigurierten Input-Konfigurationen (vgl. Abschnitt 2.3.1). Die empfangenen Daten werden in einer Elasticsearch Suchmaschine hinterlegt und nach vom Administrator definierten Filterausdrücken durchsucht. Hierbei können bestimmte Parameter mit regulären Ausdrücken aus den empfangenen Daten in eigenen Attributen hinterlegt werden, wie in Abbildung 2.3 zu sehen. Die erfassten Nachrichten sind im Anschluss inkl. der oben genannten Attribute über die Weboberfläche und die REST-API durchsuchbar. Für die Suche wird eine an Apache Lucene (Programmbibliothek für Volltextsuche) angelehnte Syntax verwendet. Diese ermöglicht die Entwicklung komplexer Suchbegriffe. Ein möglicher Suchbegriff für den ganzzahligen Parameter `http_response_code` aus der Abbildung 2.3 im Bereich 400 bis 499 wäre beispielsweise `http_response_code:[400 TO 499]`.

Abbildung 2.3.: Extrahieren von Daten aus eingehenden Meldungen in Graylog.

2.3.3. Zugriff per API

Graylog stellt eine REST-API zur Verfügung, auf welche sowohl mit der integrierten Webschnittstelle als auch programmgesteuert aus der Ferne zugegriffen werden kann. Die integrierte Webschnittstelle nutzt die API für das Beziehen der Informationen aus der Elasticsearch Suchmaschine. Jede Graylog-Instanz bietet standardmäßig eine interaktive Oberfläche an, über welche auf die Dokumentation der verfügbaren Funktionen zugegriffen werden kann. Es ist ebenfalls möglich, die API-Funktionen aus dem Browser heraus mit selbstgewählten Parametern aufzurufen. Dies erleichtert die Softwareentwicklung, da so keine externen Werkzeuge für den Zugriff auf die API verwendet werden müssen.

2.4. Amazon Web Services

Der Cloudanbieter AWS ist ein Tochterunternehmen des US-amerikanischen Versandhändlers Amazon. Das Unternehmen bietet seine Dienste vor allem für professionelle Endanwender und Unternehmen an. AWS bietet eine Vielzahl von Diensten, welche on-demand (dt. auf Abruf) verfügbar sind, nach dem Prinzip ‚pay as you go' (dt. ‚abgerechnet nach Verbrauch') berechnet werden. Zu den bekanntesten Diensten zählen EC2 (elastic compute cloud), welcher virtuelle Maschinen bereitstellt, sowie S3 (simple storage service), für das Buchen von Speicherkapazität. AWS gehört neben den allesamt US-amerikanischen Anbietern Microsoft Azure, Google Cloud, Oracle Cloud sowie dem chinesischen Anbieter Alibaba Cloud laut dem Marktforschungsunternehmen Gartner zu den führenden Cloudanbietern weltweit[12]. Derzeit gibt es keinen gleichwertigen Anbieter aus Europa. IONOS und Strato aus Deutschland sowie OVH aus Frankreich bieten nur einen Bruchteil der Umfänge der Konkurrenten an.

AWS betreibt weltweit zahlreiche Rechenzentren weltweit und stellt einen Großteil der Dienste an allen Standorten zur Verfügung. Zahlreiche Dienste lassen sich über die Webschnittstelle ‚AWS Konsole' verwalten. Als on-demand Anbieter steht für sämtliche Dienste auch die Bedienung über eine API-Softwareschnittstelle zur Verfügung, sodass Ressourcen bei Bedarf von Programmen selbstständig gebucht und vollständig verwaltet werden können. Die Abrechnung variiert je nach Dienst. Rechenressourcen werden nach Stunden oder Minuten abgerechnet, Speicherressourcen nach Datenmenge und auftragsbasierte Dienste (vgl. Abschnitt 2.4.1) nach Auftragskontingenten. AWS bietet zusätzlich ein freies Kontingent an. Die anfallenden Kosten lassen sich im Vorhinein mit dem AWS Pricing Calculator[13] bestimmen.

Für die Verwendung der AWS API mit der Programmiersprache Python stellt ein Entwicklerteam das offizielle SDK unter dem Namen Boto3 zur Verfügung. Das SDK erleichtert die Bedienung der verschiedenen API-Funktionen in Verbindung mit der Programmiersprache Python. Es wird mit von einzelnen API-Anfragen entkoppelten Objekten gearbeitet[14], damit entfällt die eigenständige Kommunikation mit der API durch die implementierte Software (beispielsweise mit der Programmbibliothek Requests) vollständig. Das SDK übernimmt weiterhin die Authentifizierung gegenüber der API. Ein Objekt repräsentiert jeweils einen AWS-Dienst (siehe folgende Unterabschnitte) in der Softwareumgebung und ermöglicht eine einfache Einbindung in die Software. Die verfügbaren API-Funktionen können als Methoden des Objekts in Python direkt verwendet werden.

2.4.1. Amazon Transcribe

Mit dem Dienst Amazon Transcribe können Audiodateien in Text mittels künstlicher Intelligenz transkribiert werden. Der Dienst unterstützt mehrere Sprachen und Dialekte. Standardmäßig wird ein allgemeines Sprachmodell des Anbieters verwendet. Es ist auch möglich, ein eigenes Sprachmodell zu trainieren und importieren. Bei der Bedienung über die AWS Konsole muss ein URI zu einer Audiodatei in einem S3-Speicher angegeben werden. Im Bezug auf S3-Speicher unterscheiden sich URI und URL voneinander. Die Unterschiede werden im nächsten Unterkapitel

[12] `https://www.gartner.com/technology/media-products/reprints/AWS/1-271W1OSP-DEU.html`
[13] `https://calculator.aws/`
[14] `https://boto3.amazonaws.com/v1/documentation/api/latest/guide/resources.html`

erläutert. Der Dienst hinterlegt den Text in einer Datei ebenfalls in einem S3-Speicher, optional kann dabei der Quellspeicher eingestellt werden.

Eine Übersetzung in Echtzeit ist nicht möglich. Das Limit für gleichzeitige Vorgänge pro Benutzer liegt bei 100. Neue Aufträge können so lange nicht eingereicht werden, bis die Anzahl wieder unter dem Limit liegt. Bei einer absehbaren Überschreitung des Limits kann eine Auftragswarteschlange verwendet werden, um die Aufträge zuzuführen.

Die Abrechnung erfolgt außerhalb des freien Kontingents nach Zeitkontingenten und beginnt bei 0,024 USD pro angefangene Minute.

Unterschied von URI und URL

Ein URL (engl. ‚uniform resource locator') gibt den Weg zu einer Ressource an. Ein URI (engl. ‚uniform resource identifier') gibt nicht unbedingt den Weg zu einer Ressource an, aber dessen Identifikationsmerkmal (beispielsweise ein vergebener Name). S3-URIs enthalten den Dateipfad einer Datei in einem S3-Speicher in jeweils einer geographischen Region (Beispielsweise am AWS Standort ‚eu-central-1' in Frankfurt). In anderen Regionen können gleichnamige S3-Speicher erstellt werden. Um zwischen diesen Speichern zu unterscheiden, wird ein URL benötigt, da dieser den öffentlichen Ort des Dokuments angibt. Ein Beispiel für eine S3-URI ist `s3://ba-telegram-graylog-bot/audio/tts001.mp3`. Ein Beispiel für eine S3-URL zu derselben Ressource am Standort ‚eu-central-1' ist `https://ba-telegram-graylog-bot.s3.eu-central-1.amazonaws.com/audio/tts001.mp3`.

2.4.2. Amazon Polly

Polly ist ein TTS-Dienst (Abkürzung für ‚text-to-speech', dt. ‚Text zu Sprache') und bildet das Gegenstück zu Amazon Transcribe. Der Dienst unterstützt ebenfalls mehrere Sprachen und Dialekte. Je nach Sprache können verschiedene Modelle verwendet werden, welche verschiedene Persönlichkeiten und Geschlechter darstellen. Dabei wird zwischen den Typen ‚standard' und ‚neural' unterschieden. ‚Neural'-Modelle erzeugen eine gegenüber dem Typ ‚standard' optimierte Ausgabe, welche der menschlichen Aussprache so ähnlich wie möglich kommen soll.

Die Abrechnung erfolgt außerhalb des freien Kontingents nach Zeichenkontingenten und beginnt bei 4 USD für eine Million Zeichen. Der Typ ‚neural' kostet bei gleicher Verwendung etwa das Vierfache.

2.5. Verwandte Arbeiten

Im Folgenden werden wissenschaftliche Arbeiten mit ähnlichen Zielsetzungen vorgestellt und die Unterschiede zu dieser Arbeit erläutert.

2.5.1. Voice-controlled order system

Die Abschlussarbeit von David Höijer und Hannes Jansson von der staatlichen Hochschule Halmstad in Schweden mit dem Titel „Voice-controlled order system“ und veröffentlicht am 14.06.2021 behandelt die Planung und Implementierung eines Sprachassistenten für die Aufgabe von Bestellungen bei Lieferdiensten für Lebensmittel[15]. Für die Spracherkennung und Verarbeitung der eingegangenen Aufträge wird auf Dienste der Cloudanbieter Amazon Webservices und Google Cloud zurückgegriffen. Die Unterschiede in der Umsetzung zur vorliegenden Arbeit bestehen darin, dass David Höijer und Hannes Jansson sich für eine Spracherkennung mit künstlicher Intelligenz durch Google Dialogflow und damit gegen ein statisches Modell, wie es in dieser Arbeit eingesetzt wird, entschieden haben.

2.5.2. Chatbot capable of information search

Michal Ďurista entwickelte im Rahmen seiner Bachelorarbeit an der technischen Universität Brünn in Tschechien aus dem Jahr 2019 einen Chatbot, welcher Benutzern Informationen von einer Webseite auf Abruf zur Verfügung stellt[16]. Die Benutzer kontaktieren den Bot dabei in Alltagssprache. Mithilfe von künstlicher Intelligenz analysiert der Bot die Nachrichtenbestandteile und ermittelt so die gewünschten Informationen. Der Autor erwähnt in der Arbeit Aspekte der Verarbeitung natürlicher Sprache und verwendet für die Implementierung Dienste von Microsoft Azure. Der Unterschied zur vorliegenden Arbeit besteht darin, dass die Informationen nicht maschinenlesbar vorliegen. Die Software bestimmt zentrale Schlüsselwörter der Anfragen und verwendet diese für eine oder mehrere Volltextsuchen über die vorliegenden Informationen. Die Auskünfte des Bots beschränken sich dabei auf Produktinformationen zu technischen Bauteilen eines einzelnen Unternehmens.

[15] http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-45033
[16] https://www.fit.vut.cz/study/thesis/21921/

3. Planung

In diesem Kapitel werden Anforderungen an die Software und die Systemumgebung definiert sowie verwendete organisatorische Konzepte erläutert.

3.1. Auswahl der Programmiersprache

Die Software soll in der Skriptsprache Python entwickelt werden. Für die Entscheidung sprechen mehrere Vorteile: die Plattformunabhängigkeit wird durch die Verfügbarkeit von Interpretern sowohl für unix-artige Betriebssysteme als auch Windows ermöglicht. Die Software liegt jederzeit im Quellcode vor und kann so einfach gewartet und erweitert werden. Des Weiteren existiert aufgrund der hohen Popularität[1] eine reichhaltige Auswahl an Bibliotheken und Onlinedokumentationen, welche die Entwicklung der Software vereinfachen.

3.2. Anforderungen

Die Software soll auf den nicht-interaktiven Betrieb als Dienst auf unix-artigen Betriebssystemen ausgelegt werden. Dies hat u. a. Auswirkungen auf die geplanten Benutzerschnittstellen. Das Programm soll zur Laufzeit keine Konsoleneingaben verlangen, da diese nur bei einem interaktiven Betrieb, z.B. in der Shell, vorgenommen werden können. Stattdessen soll mit dem Anwender vollständig über den Telegram-Messenger interagiert werden und administrative Einstellungen sollen über Konfigurationsdateien vorgenommen werden können.

Zur Anwendung von geänderten Einstellungen ist unter Umständen ein Neustart der Software notwendig. Die Software sollte daher zustandslos arbeiten, um einen möglichen Verlust von eingegangenen und noch nicht verarbeiteten Daten zu verhindern. Weiterhin sollte die durch einen Neustart verursachte Ausfallzeit so gering wie möglich gehalten werden.

Die Software soll Ereignismeldungen über die Standardausgabe sowie über eine textbasierte Protokolldatei ausgeben, um Fehleranalysen zu vereinfachen. Es soll möglich sein, den Bot von mehreren Geräten gleichzeitig zu kontaktieren. Der Bot muss die Autorisierung von Benutzern prüfen, damit unerwünschter Datenabfluss verhindert wird.

3.2.1. Systemumgebung

Für den Betrieb der Software ist eine vorinstallierte Python 3 Umgebung notwendig. Zum Zeitpunkt der Entwicklung wurde die Version 3.9 verwendet. Die benötigten Bibliotheken sollen

[1] `https://www.tiobe.com/tiobe-index/python/`

über den Paketmanager Pip bezogen und aktuell gehalten werden können. Auch wenn die Software auf den Betrieb als Dienst ausgelegt wird, soll eine interaktive Verwendung mit unix-artigen Betriebssystemen möglich sein. Außerdem ist eine Internetverbindung notwendig, um die Server der HTTP-APIs von Telegram und AWS zu erreichen. Graylog Open muss bereits installiert und an die zu überwachenden Systeme angeschlossen sein. Die Systemanforderungen entsprechen denen des verwendeten Betriebssystems. Da sämtliche Verbindungen zu externen Systemen via HTTP(S)-Verbindungen hergestellt werden, gibt es keine weiteren Anforderungen für den Betrieb der Software. Für eine Minimalkonfiguration kann die Software in vollem Funktionsumfang bereits beim Betrieb in einer Softwareentwicklungsumgebung, z.B. auf einem Notebook verwendet werden. Für den späteren Produktivbetrieb eignet sich ein System, welches günstig dauerhaft betrieben werden kann, beispielsweise eine AWS EC2-Instanz.

3.3. Programmablauf

3.3.1. Selbsttest

Nach dem Start der Software muss geprüft werden, ob ein einwandfreier Betrieb möglich ist. Dazu ist es notwendig, die Erreichbarkeit und Funktion sämtlicher Dienste mittels geeigneter API-Abfragen zu prüfen. Nachdem die Vorbereitungen abgeschlossen sind, kann in den Regelbetrieb übergegangen werden, in welchem sich das Programm bis zum Programmende befindet.

3.3.2. Regelbetrieb

Im Regelbetrieb reagiert die Software in Echtzeit auf eingehende Nachrichten vom Anwender. Um den Betrieb der Software auch mit nicht-öffentlichen IP-Adressen (beispielsweise in einem Heimnetzwerk hinter einem NAT-Router) oder einer Firewall, welche den Zugriff auf Geräte im internen Netzwerk aus dem Internet verbietet, zu ermöglichen, soll das Verfahren des Long-polling für den Abruf von Informationen von der Telegram API verwendet werden (vgl. erstes Unterkapitel von Abschnitt 2.2.1). Somit werden lediglich Verbindungen aus dem internen Netzwerk ins Internet aufgebaut. Eventuelle NAT-Router oder Firewalls hinterlegen den Verbindungsaufbau in internen Datenstrukturen wie NAT-Tabellen und lassen Antworten aus dem Internet zu.

Der geplante Ablauf des Regelbetriebs ist im Sequenzdiagramm in Abbildung 3.2 abgebildet.

3.3.3. Spracherkennung

Bei der Ausarbeitung eines Konzepts für die Funktionsweise der Spracherkennung müssen einige funktionale und nicht-funktionale Eigenschaften beachtet werden. Die Entscheidung wird u. a. beeinflusst durch Aspekte der ...

- Erweiterbarkeit: es soll einfach und insbesondere ohne ein notwendiges Training von Sprachmodellen möglich sein, neue Systeme und Abfragen zu definieren.
- Fehlertoleranz: die Aussprache einzelner Wörter verändert sich durch grammatikalische Eigenschaften je nach Satzbau leicht. Die Erkennung muss trotz dieser Veränderungen zuverlässig funktionieren.

- Rechenkapazität: die Geschwindigkeit (Wartezeit während des Vorgangs) der Spracherkennung sollte nicht von begrenzten Rechenressourcen oder der Länge der zu übersetzenden Nachricht abhängen.
- finanziellen Kosten: diese sollten möglichst gering gehalten werden.
- Plattformunabhängigkeit: die Software und ggf. trainierte Modelle sollten wie die gewählte Programmiersprache Python plattformunabhängig und portabel sein.
- Nachvollziehbarkeit: während der Entwicklung und bei der Bedienung sollten auftretende Fehler einfach erkenn- und behebbar sein.
- Sprache: das System soll Sätze verstehen, welche Begriffe aus mehreren Sprachen (Deutsch und Englisch) beinhalten.

Es bestehen verschiedene Möglichkeiten, Sprache zu Text zu transkribieren und den Inhalt im Anschluss zu analysieren, um das Anliegen des Anwenders zu erkennen und eine passende Anfrage für die Suchmaschine in Graylog zu bilden. Für die Transkription ist es notwendig, künstliche Intelligenz einzusetzen. Diese kann lokal oder entfernt ausgeführt werden. Die entfernte Ausführung bietet Vorteile bezüglich der Plattformunabhängigkeit und der (von den lokalen Ressourcen unabhängigen) Geschwindigkeit. Es stehen verschiedene Online-Dienste für die Transkription zur Verfügung, darunter Watson Speech to Text von IBM, Amazon Transcribe von AWS und Speech-To-Text vom Anbieter Google Cloud.

Syntax

Liegt die eingegangene Nachricht als Text vor, müssen die Inhalte analysiert werden. Der Aufbau der Nachricht muss einem festen Muster folgen, welches durch eine Prozedur analysiert werden kann. Hierzu eignet sich die Verwendung von Schlüsselwörtern. Diese bieten den Vorteil, dass keine Analyse der Grammatik notwendig ist. Die Verwendung von künstlicher Intelligenz bietet hierbei keine Vorteile. Das KI-Modell müsste für neue Begriffsdefinitionen, Grammatik (Satzbau) sowie Umgangssprache trainiert werden. Dies ist bei der geplanten Anwendung des Programms nicht durchführbar, da der Aufwand für das Hinzufügen neuer Suchbegriffe möglichst gering gehalten werden soll. Weiterhin bietet ein Algorithmus für die Analyse einer Nachricht mit statischem Aufbau den Vorteil der besseren Nachvollziehbarkeit des ermittelten Ergebnisses.

Es wird ein Aufbau aus Produktkategorie, Eigenschaft und Zeitraum gewählt. Die Produktkategorie entspricht der abzufragenden Gerätegruppe, beispielsweise ‚Webserver'. Für jede Produktkategorie können Eigenschaften definiert werden, für die Abfrage von Ereignissen mit einem Statuscode `4xx` oder `5xx` der Gruppe ‚Webserver' beispielsweise ‚Fehlermeldungen'. Ein weiteres Schlüsselwort führt zu den Informationen für den abzufragenden Zeitraum, welcher relativ angegeben wird (‚letzte fünf Tage', ‚letzte Woche', ‚letzte 20 Minuten').

Vergleich diverser Transkriptionsdienste

Die Umwandlungsgenauigkeit der in Abschnitt 3.3.3 genannten Dienste soll mit einem Testset bestehend aus umzuwandelnden Audiodateien mit möglichen Sprachanfragen geprüft werden. Dazu wurden zehn Sprachnachrichten über die Telegram App für iOS auf einem iPhone aufgenommen. Im Anschluss wurden die Audiodaten von der Telegram API extrahiert, sodass diese

im OGG-Format vorlagen. Die drei Dienste bevorzugen jeweils andere Audioformate, daher wurden die Dateien aus dem OGG-Format (für das Produkt von Google Cloud) in das MP3- (für das Produkt von AWS) und das FLAC-Format (für das Produkt von IBM) umgewandelt. Schließlich wurden jeweils zehn Audiodateien über eine Webkonsole, welche von allen Anbietern angeboten wurde[2], an den Dienst übermittelt. Die Webkonsolen verwendeten jeweils die APIs, welche ebenfalls für den programmgesteuerten Zugriff verwendet werden. Die Ergebnisse werden in der Tabelle 3.1 abgebildet.

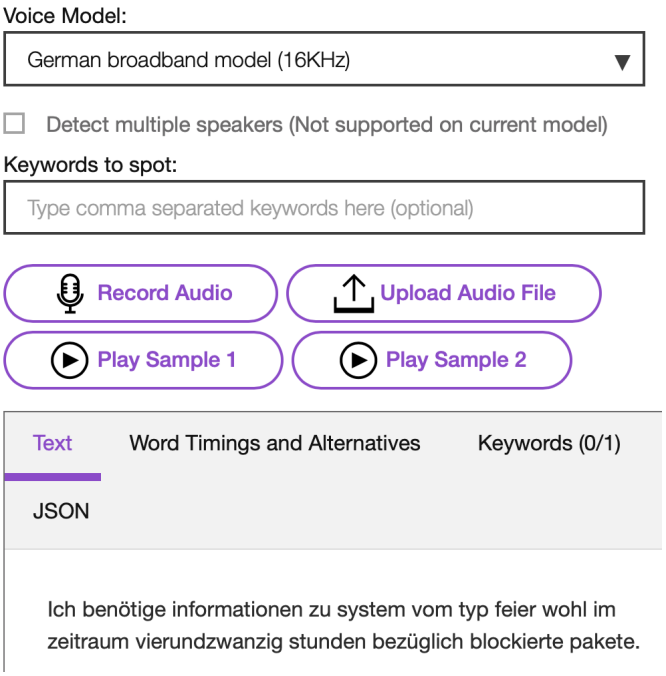

Abbildung 3.1.: Eingabemaske von IBM Watson Speech to Text nach Bearbeitung von Testset06 mit einer unzutreffenden Ausgabe.

Die Audionachrichten beinhalten folgende gesprochene Texte:

- *Testset01*: „Typ Webserver Eigenschaft Fehler Zeitraum 24 Stunden“
- *Testset02*: „Typ Webserver bezüglich Erreichbarkeit Zeitraum 24 Stunden“
- *Testset03*: „Typ Webserver bezüglich Besucherzugriffe Zeitraum 24 Stunden“
- *Testset04*: „Typ Webserver bezüglich Besucherzugriffe Zeitraum 24 Stunden“
- *Testset05*: „Ich benötige Informationen zu Systemen vom Typ Firewall im Zeitraum 24 Stunden bezüglich blockierte Pakete“
- *Testset06*: „Typ eins bezüglich drei Zeitraum ein Tag“
- *Testset07*: „Typ eins bezüglich drei Zeitraum ein Tag“
- *Testset08*: „Typ Webserver Zeitraum 24 Stunden bezüglich Erreichbarkeit“
- *Testset09*: „Typ Webserver Zeitraum 24 Stunden bezüglich Erreichbarkeit“
- *Testset10*: „Typ Mailserver bezüglich empfangene Mails Zeitraum ein Jahr“

[2] `https://eu-central-1.console.aws.amazon.com/transcribe/home?region=eu-central-1#createJob`, `https://speech-to-text-demo.ng.bluemix.net`, `https://cloud.google.com/speech-to-text`

Die Transkriptionsdienste von AWS und Google Cloud konnten durch eine geringe Fehlerquote gegenüber dem Produkt vom IBM überzeugen. Aufgrund von bereits existierenden persönlichen Erfahrungen in der Verwendung von AWS wird Amazon Transcribe als Transkriptionsdienst verwendet.

Konfiguration des Bots

Die Erweiterung der Software soll einfach möglich sein. Produktgruppen sollten darüber hinaus dynamisch erweiterbar sein: Wird eine Abfrage für die Produktkategorie ‚Webserver' getätigt, sollen alle derzeit an Graylog angeschlossenen Webserver in die Suche einbezogen werden.

Die Software ist für die Bedienung durch Systemadministratoren vorgesehen. Daher soll für die Konfiguration der Abfragen eine Textdatei verwendet werden. Diese ermöglicht eine schnelle Anpassung der Konfiguration und bietet gleichzeitig einen Überblick über bestehende Suchbegriffe. Für die einfache Verwendung mit einer Python-Bibliothek stehen mehrere Dateiformate zur Verfügung, darunter JSON, INI, TOML, YAML und XML. Das YAML- und XML-Format ist für die Verwaltung über einen Texteditor ohne Syntaxhervorhebung und Funktionen wie automatischem Einrücken nicht geeignet.

Das TOML-Format vereint die Vorteile der einfachen Lesbarkeit des INI-Formats mit den Vorteilen der Verwendung von verschiedenen Datentypen des JSON-Formats. TOML entspricht weitestgehend der INI-Syntax. Ein Unterschied besteht darin, dass von der Spezifikation verschiedene Datentypen unterstützt werden[3], darunter Zeichenketten (engl. Strings). Diese können im Falle von enthaltenen Leerzeichen in Anführungszeichen geschrieben werden. Die Hierarchie innerhalb einer TOML-Datei kann mit dem im vorherigen Unterkapitel „Syntax" beschriebenen Aufbau abgebildet werden. TOML-Sektionen werden mit eckigen Klammern markiert und entsprechen den Produktkategorien (vgl. zweite Zeile des Listing 3.1). Sie beinhalten Name-Wert-Paare, welche durch ein Gleichheitszeichen getrennt werden (vgl. dritte Zeile des Listing 3.1). Ein Name darf in einer Sektion nur jeweils einmal vorkommen, eine Sektion darf in einer Datei nur jeweils

[3] `https://toml.io/en/v1.0.0`

Testset	AWS	Google	IBM
01	-	+	+
02	+	+	-
03	+	+	+
04	-	-	-
05	+	+	-
06	+	+	-
07	+	+	-
08	+	+	-
09	+	-	-
10	+	+	-
Summe Fehler	2	2	8

Tabelle 3.1.: Ergebnisse der Transkriptionsaufträge.

einmal vorkommen. Sektionen, Name und Werte dürfen Leerzeichen enthalten. Name und Werte werden in diesem Fall in Anführungszeichen gesetzt und so als String definiert. Die im vorherigen Unterkapitel „Syntax“ definierten Eigenschaften von Produktkategorien werden durch die Namen in der TOML-Syntax repräsentiert. Die mit den Eigenschaften verknüpften Suchbegriffe werden durch die Werte der Name-Wert Paare in der TOML-Syntax repräsentiert. Zusätzlich ist es möglich, einzeilige Kommentare mit `#` einzuleiten.

Listing 3.1: Beispiel der TOML-Syntax.

```
1 # Dies ist ein Kommentar
2 [Sektion]
3 Name = "Wert"
4
5 [Webserver]
6 Fehler = http_response_code:[400 TO 599]
```

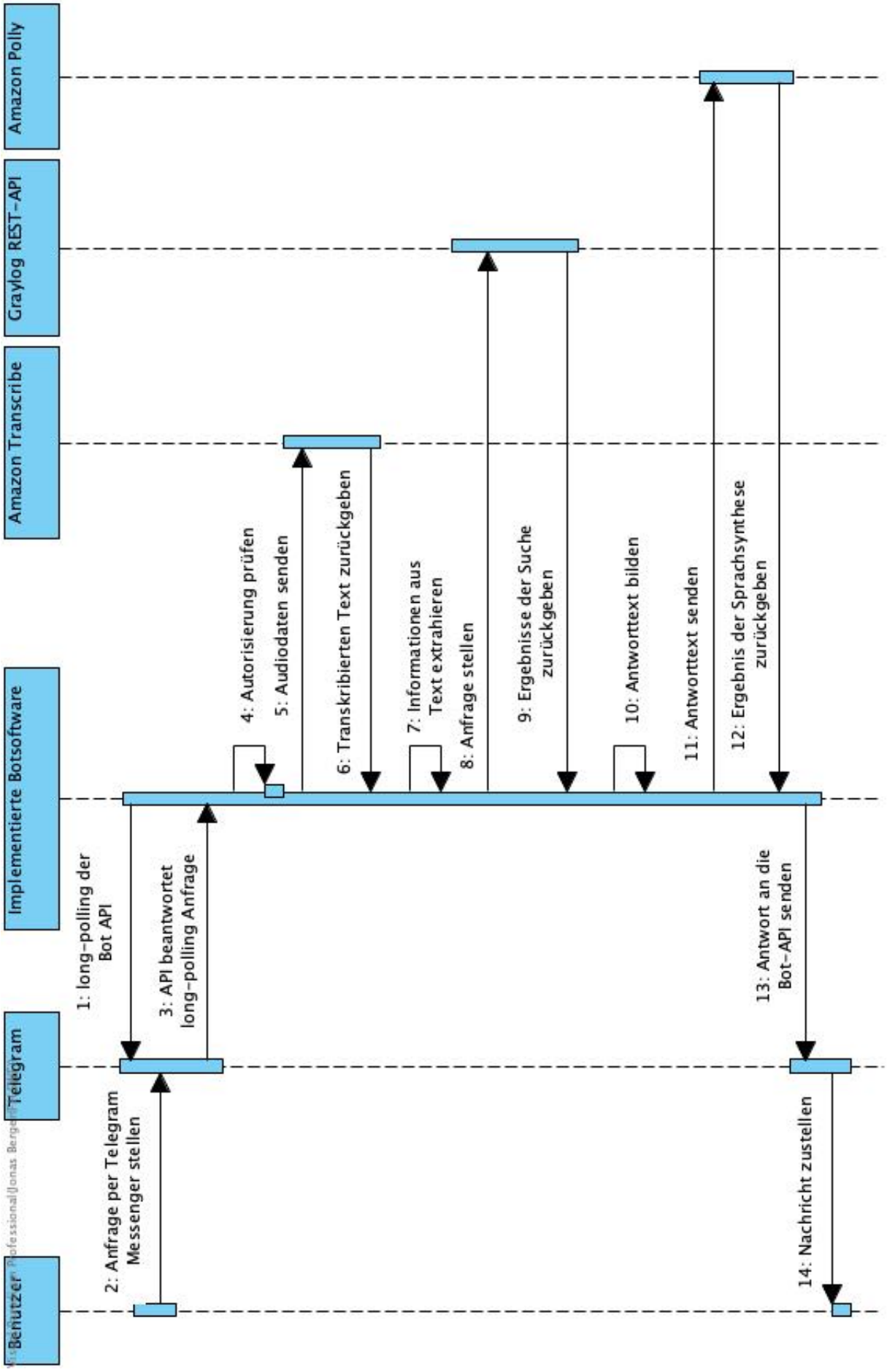

Abbildung 3.2.: Sequenzdiagramm für Regelbetrieb.

4. Implementierung

4.1. Struktur

Um eine modulare Implementierung der Software zu ermöglichen und darüber hinaus den Quellcode getrennt von nicht für die Öffentlichkeit bestimmten Daten (beispielsweise Anmeldedaten für APIs) bereitstellen zu können, ist es notwendig, die Softwareentwicklungsumgebung entsprechend zu gestalten. Die Unterteilung der für den Betrieb notwendigen Dateien inkl. des Quellcodes wird in den folgenden beiden Unterkapiteln beschrieben. Um die Lesbarkeit und Erweiterbarkeit zu verbessern, wurden die im Python Enhancement Proposal (PEP) Nr. 8[1] vorgeschlagenen Formatierungsrichtlinien umgesetzt. Sämtliche Module und Funktionen enthalten nach der Definition einen mehrzeiligen Kommentar mit einer Docstring-Dokumentation gemäß PEP 257[2] im Quellcode, siehe Listing 4.1.

Listing 4.1: Beispiel für einen Docstring.

```
def execute_query(query, seconds):
    """
    Diese Funktion führt eine ElasticSearch-Abfrage über die Graylog API
        aus.
    :param query: str, Graylog ElasticSearch-Abfrage
    :param seconds: int, Anzahl der Sekunden des relativen Zeitraums
    :return: Gibt die Anzahl der gezählten Ereignisse zurück
    """
```

Als Docstring wird ein mehrzeiliger Kommentar bezeichnet, welcher in der Zeile nach der Funktionsdefinition für den Entwickler wichtige Informationen enthält. Durch die standardisierte Syntax können die Informationen in vielen IDEs, darunter PyCharm von JetBrains, für eine interaktive Dokumentation innerhalb der Entwicklungsumgebung verwendet werden. Der Docstring enthält einen kurzen Satz zum funktionalen Umfang. Im Anschluss werden die Parameter und die von der Funktion zurückgegebenen Daten erläutert. Da Python dynamische Typisierung anwendet (Variablen werden nicht explizit mit einem Datentyp wie Ganzzahl oder Zeichenkette initialisiert sondern der Datentyp hängt vom zugewiesenen Wert ab), sorgen die Angaben für eine beschleunigte und, durch die Vermeidung von Typkonflikten, weniger fehleranfällige Entwicklung.

4.1.1. Dateisystem

Nachfolgend wird die Struktur der in Abbildung 4.1 dargestellten Ordner und Dateien erläutert.

[1] https://peps.python.org/pep-0008/
[2] https://peps.python.org/pep-0257/

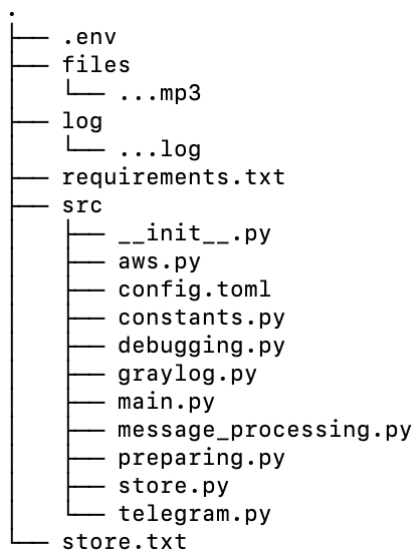

Abbildung 4.1.: Gekürzte Ausgabe der für den Betrieb notwendigen Ordner und Dateien im Basisverzeichnis.

Von oben nach unten:

- ‚.env' beinhaltet sensible Zugangsdaten für die verschiedenen Dienste und wird nicht in das öffentliche GitHub-Repository hochgeladen.
- ‚files' dient als Zwischenspeicher für anfallende Audiodateien, welche aus den eingehenden Sprachnachrichten extrahiert wurden oder per Sprachsynthese erzeugt wurden.
- ‚log' enthält die textbasierten Protokolle der Software. Zu jedem Start der Software wird eine neue Datei mit dem aktuellen Datum erzeugt. Die Inhalte der Dateien entsprechen der Konsolenausgabe mit der Ausnahme, dass sensible Daten nur in der Konsolenausgabe ausgegeben werden.
- ‚requirements.txt' dient als Informationsquelle für den Paketmanager Pip, mit welchem die Abhängigkeiten aus dem Python Package Index installiert werden können. Die Datei beinhaltet Namen und Versionsnummern von externen Softwaremodulen.
- ‚src' beinhaltet Dateien mit dem Quellcode.
- ‚___init___.py' ist notwendig für die Moduldefinitionen in Python und hat keinen Inhalt
- ‚store.txt' wird für die Persistierung des Werts für den Parameter ‚update_id' der Telegram Bot API verwendet.[3, S. 337].

  Weitere Dateien im Quellcode-Verzeichnis entsprechen den Modulen und werden im folgenden Abschnitt 4.1.2 detaillierter erläutert.

4.1.2. Module

Der Quellcode wurde in mehrere Module aufgeteilt:

- `main` enthält die Hauptfunktion des Programms und startet die Vorbereitung und den Übergang in den Regelbetrieb.

- `debugging` enthält die für die Fehlersuche notwendigen Funktionen. Siehe Abschnitt 4.2.
- `preparing` enthält die für den Startvorgang notwendigen Funktionen, welche nicht Teil eines anderen Moduls sind. Siehe Abschnitt 4.3.
- `telegram` enthält Funktionen für den Zugriff auf die Telegram-API. Die HTTP-Abfragen werden mit der Programmbibliothek Requests[3] durchgeführt. Weiterhin sind Funktionen für die Bearbeitung der Antworten der API enthalten. Die Authentifizierung an der API erfolgt über nur dem Entwickler bekannte Zugriffsschlüssel, welche vom Telegram-Bot `@BotFather` ausgestellt werden.
- `aws` enthält Funktionen für den Zugriff auf AWS. Für die Kommunikation mit der API wird Boto3, das vom AWS-Team für Python bereitgestellte SDK, verwendet.
- `constants` beinhaltet Konstanten, mit welchen der Ablauf des Programms gesteuert werden kann (beispielsweise können detaillierte Meldungen zum Programmablauf aktiviert oder bestimmte Telegram-Benutzer für den Zugriff auf den Bot autorisiert werden). Des Weiteren sind Variablen als Platzhalter Teil des Moduls, welche zur Laufzeit mit konstanten Objektdefinitionen der Boto3-Bibliothek überschrieben werden. Die Objektdefinitionen ermöglichen einen abstrakten Zugriff auf die AWS API, beispielsweise ohne notwendige erneute Authentifizierungen (dies übernimmt Boto3 im Hintergrund).
- `graylog` beinhaltet Funktionen für den Zugriff auf die Graylog API. Der Zugriff erfolgt ähnlich wie beim Modul `telegram` mit der Programmbibliothek `requests`. Aus Sicherheitsgründen werden verschiedene Zugangsschlüssel für die Autorisierung und Authentifizierung verwendet[4]. Der Access Token (dt. Zugriffsschlüssel) entspricht den Anmeldedaten an der Graylog Webschnittstelle und ermöglicht einen dauerhaften Zugriff. Mit dem Access Token können zeitlich befristete Session Tokens (dt. Sitzungsschlüssel) generiert werden. Die Software erkennt anhand der von der API zurückgegebenen Fehlermeldung automatisch, wenn ein nicht gültiger Session Token verwendet wurde, und beantragt in diesem Fall einen neuen.
- `message_processing` enthält interne Funktionen, welche für die Verarbeitung der erhaltenen Nachrichtentexte vorgesehen sind.
- `store` enthält Funktionen, welche für die Persistierung von Informationen notwendig sind. Die Telegram-API verwendet einen Zähler, welcher für jede Aktualisierung um einen Schritt inkrementiert wird. Ruft der Client Aktualisierungen von der API ab, erhält er den derzeitigen Stand als Wert der Variable ‚update_id' (vgl. fünfte Zeile aus Listing 2.2). Bei weiteren Anfragen an die API gibt der Client die Variable ‚update_id' an, um nur Aktualisierungen zu erhalten, die seit der letzten Abfrage eingetroffen sind (und daher einen höheren Wert der Variable ‚update_id' haben). Die Variable ‚update_id' wird nach jeder Abfrage in die Textdatei store.txt auf das lokale Dateisystem geschrieben.

[3] `https://pypi.org/project/requests/`
[4] `https://docs.graylog.org/docs/rest-api`

4.2. Protokollierung

Für die Protokollierung wurde ein bereits vorhandenes, vom Autor dieser Arbeit entwickeltes Programmmodul erweitert, das Modul ‚debugging' des GitHub-Repositorys ‚Jomibe/wireguard-config-generator'[5]. Das Repository enthält den Quellcode sowie die Dokumentation eines Softwareprojekts, welches im Rahmen des Moduls ‚Projekt' im Sommersemester 2022 an der Fachhochschule Südwestfalen implementiert wurde. Es wurde eine Software entwickelt, mit welcher die Konfigurationsdateien eines WireGuard-VPN-Servers verwaltet werden können. Das Programm ermöglicht es unter anderem, die existierenden Konfigurationsdateien auf Syntax- und semantische Fehler zu prüfen und Konfigurationen für weitere VPN-Clients inkl. der Bestimmung von geeigneten Netzwerkkonfigurationen hinzuzufügen. Das Modul ‚debugging' enthält Funktionen, welche für die Fehlersuche während der Laufzeit verwendet werden. In dem Modul wird die Funktion `console()` implementiert, um die Ausgabe von Statusmeldungen auf der Konsole zentral zu koordinieren.

Alle Statusmeldungen werden in einer Textdatei hinterlegt, deren Pfad in der Datei ‚config.py' konfiguriert wird. Ist der `DEBUG`-Modus aktiv, erscheinen alle Meldungen zusätzlich auf der Konsole. Ist die Fehlermeldung mit dem Parameter `secret` gekennzeichnet, erfolgt keine Protokollierung in der Textdatei. Werden Meldungen auf der Konsole ausgegeben, werden diese gemäß dem Schweregrad farblich markiert. Dazu wird das die Programmbibliothek Colorama[6] verwendet.

Bei der Formulierung der Fehlermeldungen wurden einige Anforderungen beachtet [4, S. 495]:

- Details zu fehlenden Informationen und Hinweise, wie diese Informationen übermittelt werden können. In diesem Anwendungsfall werden beispielsweise die Schlüsselwörter für Produktkategorien, Eigenschaften und Zeiträume genannt, wenn Informationen dazu fehlen.
- Vermeidung von für den Anwender unverständliche technische Ausdrücken in den Fehlermeldungen.
- Fehlermeldungen müssen den Benutzer möglichst schnell auf die Fehlerursache hinweisen und dürfen nicht vorwurfsvoll formuliert sein.

In Abbildung 4.2 und Abbildung 4.3 ist die farbige Darstellung in der Konsole zu sehen. Im Anhang befinden sich zwei Listings mit ausführlicheren Konsolenausgaben, welche im folgenden Absatz erläutert werden.

```
Erfolg: Name für die Protolldatei: log/             .log
Info: Lese sensible Daten aus der Datei .env
Fehler: Die Datei .env enthält nicht alle erforderlichen Angaben.
Folgende Parameter müssen definiert sein: TELEGRAM_BOT_TOKEN, GRAYLOG_USERNAME
Fehler: Umgebungsvariablen konnten nicht initialisiert werden
Fehler: Der Programmablauf wird abgebrochen
```

Abbildung 4.2.: Farbige Ausgaben mit Fehlermeldungen.

Listing A.1) zeigt die Konsolenausgabe während des Startvorgangs. Zu Beginn wird die Protokollierung in eine Textdatei initialisiert. Danach werden sensible Informationen wie Zugangsdaten aus der Datei .env in Umgebungsvariablen übernommen. Im Anschluss wird die Erreichbarkeit

[5] https://github.com/Jomibe/wireguard-config-generator/blob/main/src/debugging.py
[6] https://pypi.org/project/colorama/

```
Erfolg: Verbindung zu AWS Polly aufgebaut
Info: Wiederherstellen der aktuellen update_id...
Info: Prüfe, ob die Datei store.txt existiert. Datei wird falls notwendig erstellt.
Erfolg: Die Datei existiert.
Warnung: Es ist keine gültige Update-ID in der Datei store.txt hinterlegt
Erfolg: Wiederherstellung der aktuellen update_id 1 abgeschlossen.
```

Abbildung 4.3.: Farbige Ausgaben mit Warnmeldungen.

der Dienste von Telegram, Graylog und AWS geprüft. Schließlich wird die Konfiguration aus der Datei config.toml in eine interne Datenstruktur geladen, in welcher die Aliasdefinitionen (diese Funktion wird im ersten Unterkapitel des Abschnitts 4.4.2 beschrieben) aufgelöst werden.

(Listing A.2) zeigt die Konsolenausgabe beim Verarbeiten einer eingegangenen Sprachnachricht. Im ersten Schritt wird der Zähler der Variable `update_id` erhöht. Danach wird die Audiodatei von der Telegram Bot API bezogen, auf dem lokalen Dateisystem gespeichert und zu AWS Transcribe übertragen. Nachdem der Übertragungsprozess abgeschlossen ist, werden die Zwischenergebnisse der Transkription angezeigt. Ist der Transkriptionsprozess abgeschlossen, wird der Benutzer über den Status informiert und die Nachricht wird, nun in Textform, auf Schlüsselwörter analysiert. Anhand der Schlüsselwörter werden die Informationen extrahiert und mit den Werten aus der beim Start gefüllten Datenstruktur verglichen. Es wurde eine Übereinstimmung gefunden, somit kann eine Anfrage an Graylog gestellt werden. Das Ergebnis wird dem Benutzer mitgeteilt. Die Verarbeitung ist abgeschlossen und das Programm beginnt erneut mit dem Long-polling der Telegram Bot API.

4.3. Startvorgang

Das Modul `main` enthält die Methode `main()`, welche den Startpunkt des Programms darstellt. In der Vorbereitungsphase (vgl. Abschnitt 3.3.1) wird zuerst die Programmbibliothek Colorama initialisiert. Dies ist notwendig, um die Steuerung der Farbausgabe auf der Konsole an das zur Laufzeit verwendete Betriebssystem anzupassen[7]. Nach der Initialisierung wird die Funktion `prepare()` aus dem Modul `preparing` aufgerufen. Die Funktion besteht aus weiteren Funktionsaufrufen, welche jeweils mit einer positiven Rückmeldung abgeschlossen werden müssen. Wird eine Prüfung nicht erfolgreich abgeschlossen, gibt `prepare()` den Wert `False` zurück, und der Programmablauf wird nach Ausgabe einer detaillierten Fehlermeldung abgebrochen. Falls kein Fehlerstatus zurückgegeben wird, wird zum Regelbetrieb übergegangen. Dieser besteht aus einer Endlosschleife, in welcher die Funktion `check_updates()` des Moduls `telegram` aufgerufen wird.

4.4. Regelbetrieb

Im Regelbetrieb wird durch die Hauptfunktion in einer Endlosschleife die Funktion `getUpdates()` aufgerufen (siehe Listing 4.2). Diese führt Long-Polling der API-Funktion `getUpdates` durch (siehe Listing 4.3). Eingehende Nachrichten werden in Echtzeit erkannt und weiterverarbeitet. Falls mehrere Nachrichten vorliegen (dies kommt vor, wenn die Software für längere Zeit nicht

[7] `https://github.com/tartley/colorama#initialisation`

mit der Telegram-API verbunden war und in der Zwischenzeit mehrere Nachrichten an den Bot gesendet wurden), wird jeweils nur eine Nachricht verarbeitet, da die Variable ‚update_id' nur um jeweils einen Schritt (statt um die Anzahl der neuen Nachrichten) inkrementiert wird. Der Wert für den Ablauf der HTTP-Anfrage sollte so hoch wie möglich, aber so gering wie notwendig gewählt werden. Um Probleme durch Netzwerkinfrastruktur und Serverprozesse zu vermeiden, beträgt dieser standardmäßig 30 Sekunden [5, Abs. 5.5. Timeouts] und kann mittels des Parameters `TELEGRAM_LONG_POLL_TIMEOUT` verändert werden. Dieser Parameter beeinflusst, wie lange die Verbindung seitens des Clients (diese Rolle übernimmt hier die in dieser Arbeit entwickelte Software) geöffnet bleibt. Nachdem die Zeit vergangen ist, wird die Anfrage mit einem Timeout vom Client abgebrochen. Durch die Endlosschleife wird sofort eine neue Anfrage gestartet. Trifft eine Antwort des Servers vor Ablauf der Zeitspanne ein, wird die Nachricht verarbeitet und die Software eröffnet im Anschluss eine neue Anfrage mit der angegebenen Wartezeit. Dies wird in Abbildung 4.4 dargestellt. Der Ablauf der Verarbeitung wird im folgenden Absatz beschrieben.

Listing 4.2: Programmcode der Hauptfunktion ohne Kommentare und Ausgaben zur Fehleranalyse

```
def main():
    colorama.init()
    if not prepare():
        console("Der Programmablauf wird abgebrochen", mode=ERR)
        return 1

    while True:
        check_updates()
```

Listing 4.3: Programmcode der Funktion für den Abruf von Aktualisierungen von der Telegram Bot API ohne Kommentare und Ausgaben zur Fehleranalyse

```
def get_updates():
    r = requests.get(url=f'https://api.telegram.org/bot{constants.
       telegram_bot_token}/getUpdates',
                     params={"offset": constants.telegram_update_id + 1,
                             "timeout": f"{TELEGRAM_LONG_POLL_TIMEOUT}",
                             "allowed_updates": '["message"]'
                             })
    if r.status_code == 200:
        console("Antwort in", f"{r.elapsed.microseconds/1000}ms", "
           erhalten", mode=SUCC)
    else:
        console("Telegram API ist nicht erreichbar. Details:", f"{r.
           status_code} {r.reason} - {r.text}", mode=ERR)
        return None
    return json.loads(r.text)
```

Zuerst wird anhand der Chat ID (vgl. Zeile 16 in Listing 2.2) ermittelt, ob der Benutzer durch den Administrator für die Verwendung des Bots freigegeben wurde. Hierzu wird der Wert mit der Liste `constants.AUTHORIZED_CHAT_IDS` verglichen. Bei positivem Ergebnis erfolgt die Verarbeitung der Nachrichteninhalte: Anhand des Aufbaus des JSON-Objekts wird ermittelt,

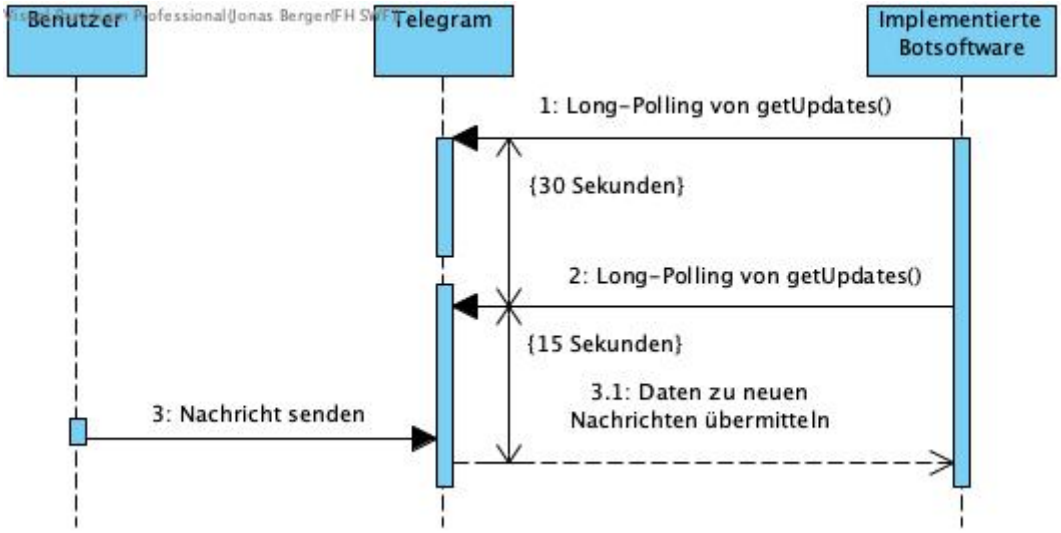

Abbildung 4.4.: Darstellung des zeitlichen Ablaufs des Long-pollings. Die erste Nachricht trifft 45 Sekunden nach Programmstart ein.

ob es sich um eine Text- oder Sprachnachricht handelt. Der Text einer Nachricht wird direkt durch die Funktion `message_processing.process_text_message` verarbeitet. Handelt es sich um eine Sprachnachricht, wird zuerst die Audiodatei von der Telegram Bot-API bezogen und auf dem lokalen Dateisystem abgelegt. Danach erfolgt der Transkriptionsprozess in einer weiteren Funktion, welche den ermittelten Text in einer Zeichenkette zurückgibt. Schließlich wird der Text ebenfalls durch die Funktion `message_processing.process_text_message` verarbeitet.

4.4.1. Optimierung der Antwortzeit

Die Bedienung eines Systems mittels Sprache erfordert eine umfangreichere Benutzerführung als die Bedienung eines textbasierten Systems über einen Bildschirm und eine Tastatur. Beim Aufruf einer Webseite bietet der Bildschirm durch die Anzeige des Webbrowsers mit diversen Statuselementen eine dauerhafte Möglichkeit für den Benutzer, zu prüfen, ob die gewünschte Anfrage eingegangen ist und verarbeitet wird. Eine nur auf Sprache basierende Bedienung bietet keine äquivalente Möglichkeit, dem Benutzer eine Statusübersicht bis zum Abschluss der Anfrage zur Verfügung zu stellen. Um Missverständnisse vorzubeugen und die Bedienung komfortabel zu gestalten, muss das System eine Rückmeldung innerhalb eines durch den Menschen als nicht zu lang empfundenen Zeitraums zurückgeben. Dazu sollte die Zeit ohne sichtbare Veränderung eine Dauer von 10 Sekunden nicht überschreiten [6, Kapitel 5.5][8].

Beim Test eines frühen Prototyps fiel auf, dass die Dauer zwischen Absenden der Sprachnachricht und Erhalt einer Antwort mit den Ergebnissen bis zu 30 Sekunden betrug. Diese Antwortzeit entspricht nicht dem oben genannten Richtwert von 10 Sekunden. Eine Analyse des Programmablaufs ergab, dass die Transkription und die Sprachsynthese (der Text-To-Speech Prozess) einen Großteil der benötigten Antwortzeit verursachten. Beide Prozesse wurden optimiert, die Optimierungen werden in den beiden folgenden Abschnitten beschrieben.

[8] Auszug aus dem Buch: `https://www.nngroup.com/articles/response-times-3-important-limits/`

Transkription

Ursprünglich wurde die Boto3-Bibliothek für die Interaktion mit Amazon Transcribe verwendet. Mit der Boto3-Bibliothek ist eine Verarbeitung in Echtzeit nicht möglich; die Audiodatei muss zuerst in einen S3-Speicher hochgeladen werden. Eine Möglichkeit für die direkte Zuführung der Datei besteht nicht. Im Anschluss muss ein Auftrag in Amazon Transcribe über das `client`-Objekt in `constants.aws_transcribe_obj` erstellt und die Datei im S3-Speicher damit verknüpft werden. Danach wird der Auftrag durch AWS verarbeitet. Die Möglichkeit eines Callbacks oder der Verwendung von Long-polling (vgl. erstes Unterkapitel von Abschnitt 2.2.1) besteht nicht, daher muss die AWS-API mittels Polling durchgehend erneut kontaktiert werden, bis der Auftrag abgeschlossen ist. Der transkribierte Text wird nach der Bearbeitung des Auftrags durch AWS in einem S3-Speicher als JSON-Objekt in einer Textdatei hinterlegt. Nachdem die Datei von S3 bezogen wurde, konnte der Text durch die Software weiterverarbeitet werden.

Die Transkription wurde optimiert, indem für den Zugriff auf Amazon Transcribe statt der Boto3-Bibliothek das Amazon Transcribe Streaming SDK[9] verwendet wird. Hierdurch ergaben sich neue Möglichkeiten für die Übermittlung der Audionachricht und des ermittelten Texts durch den Einsatz von HTTP-Streams und asynchronen Funktionen. Es wurde eine vom Hersteller bereitgestellte Vorlage für die asynchrone Verarbeitung[10] für den vorliegenden Einsatzzweck verwendet. Im Gegensatz zur oben beschriebenen Verwendung von Boto3 werden die Audiodaten hierbei direkt Amazon Transcribe über einen HTTP-Stream zugeführt. Dazu wird die zu übertragende Datei, welche sich nach dem Bezug von der Telegram Bot-API auf dem lokalen Dateisystem befindet, mittels der Python-Bibliothek Aiofile[11] in Blöcke mit einer Größe von 16 Kbyte aufgeteilt und in mehreren Paketen an die AWS-API übertragen. Bereits während der Übertragung und dem Erhalt der ersten Blöcke durch AWS beginnt die Transkription. In der Botsoftware wurde gleichzeitig ein Event Handler `handle_transcript_event()` in der Datei aws.py implementiert, welcher Daten von der AWS API in Echtzeit empfängt und nach Abschluss eines Satzes (dies ist an dem Parameter `is_partial` erkennbar) den Text der Software zuführt und die weitere Verarbeitung durch Abschluss der Funktion auslöst.

Die Entwickler des Transcribe SDK weisen darauf hin, dass sich die Software bislang in einem sehr frühen Entwicklungsstadium befindet. Zum Zeitpunkt der Entwicklung wurde die Version 0.6.0 verwendet. Um die Funktionsfähigkeit des Bots nicht durch die Abhängigkeit zum Entwicklungsstand der Transcribe SDK zu gefährden, besteht die Möglichkeit, zwischen der klassischen Transkription mit Boto3 und der Echtzeit-Transkription mit dem Konfigurationsparameter `ENABLE_FAILSAFE_TRANSCRIPTION` zu wechseln. Wird dieser auf den Wert `True` gesetzt, wird die Verarbeitungszeit der Transkription von 15 Sekunden auf etwa zwei Sekunden reduziert.

Sprachsynthese

Die Dauer des Text-To-Speech-Vorgangs wurde in ähnlicher Weise optimiert wie die der Transkription. Die dazu notwendigen Funktionen waren bereits in der Boto3-Bibliothek enthalten. Statt der Funktion `start_speech_synthesis_task` wird die Funktion `synthesize_speech` verwendet.

[9] `https://github.com/awslabs/amazon-transcribe-streaming-sdk`
[10] `https://github.com/awslabs/amazon-transcribe-streaming-sdk/blob/v0.6.0/examples/simple_file.py`
[11] `https://pypi.org/project/aiofile/`

Dadurch entfällt die Notwendigkeit, die durch AWS erzeugte Audiodatei aus einem S3-Speicher herunterladen zu müssen.

Nachdem der TTS-Auftrag gestartet wurde, muss dieser ebenfalls mittels Polling überwacht werden. Der umzuwandelnde Text wird der API weiterhin als Zeichenkette übergeben. Mit der Funktion `synthesize_speech` ist es nun möglich, die Audiodatei aus einem Stream in eine Datei auf dem lokalen Dateisystem zu schreiben.

Die Verarbeitungszeit der Sprachsynthese wurde von zehn Sekunden auf weniger als eine Sekunde verkürzt.

Effekt der Optimierungen

Die Zeit von der Erkennung neuer Nachrichten durch den Bot bis zum Abschluss des Versands der Sprachnachricht mit der Antwort beträgt nun etwa fünf Sekunden.

4.4.2. Verarbeitung des Nachrichtentexts

Die weitere Verarbeitung des Nachrichtentexts erfolgt nach dem zuvor definierten Modell (vgl. erstes Unterkapitel des Abschnitts 3.3.3) aus Produktkategorie, Eigenschaft und Zeitraum. Die Schlüsselwörter für die drei zu erkennenden Werte können über `constants.KEYWORDS*` festgelegt werden. Voreingestellt für die Produktkategorie sind ‚Typ' und ‚Kategorie', für die Eigenschaft die Schlüsselwörter ‚bezüglich' und ‚in Sachen' und für den Zeitraum ‚Zeitraum' und ‚seit'. Die Software ermittelt beim Startvorgang, wie viele Wörter die Bezeichnungen für Produktkategorie und Eigenschaft maximal umfassen. Hat eine Produktkategorie die Eigenschaften ‚Unerreichbarkeit' und ‚Interne Fehler', ist die maximale Länge der Bezeichnungen von Eigenschaften 2.

Die Texterkennung prüft nun im ersten Schritt, ob die eingegangene Nachricht jeweils ein Schlüsselwort für die Produktkategorie, Eigenschaft und den Zeitraum enthält. Im nächsten Schritt werden die auf die Schlüsselwörter folgenden Wörter für die Weiterverarbeitung erfasst. Dabei werden so viele Wörter gespeichert, wie beim Startvorgang ermittelt. Zur Erläuterung:

Listing 4.4: Beispiel für eine mögliche Konfiguration in der Datei config.toml

```
1 [Webserver]
2 "Zugriffe" = "http_response_code: 200"
3 "Interne Fehler" = "http_response_code:[500 TO 599]"
```

Die maximale Länge von Bezeichnungen der Produktkategorie ist **1**. Die maximale Länge von Bezeichnungen der Eigenschaft ist **2**.

Nachricht: „Ich benötige Informationen zu Systemen vom *Typ* Webserver *bezüglich* Zugriffe *Zeitraum* 24 Stunden"

- Produktkategorie: Schlüsselwort *Typ*, folgende **1** Wörter: Webserver
- Eigenschaft: Schlüsselwort *bezüglich*, folgende **2** Wörter: Zugriffe Zeitraum

Nun prüft die Software, ob zu den erfassten Daten Einträge in der Datei ‚config.toml' bestehen. Im obigen Fall führt Webserver durch direkte Übereinstimmung mit der ersten TOML-Sektion in Listing 4.4 zu einem Fund.

Danach werden die Eigenschaften der ermittelten Produktkategorie auf Übereinstimmung mit Zugriffe_Zeitraum verglichen. Eine solche Eigenschaft existiert nicht. Besteht der erfasste Wert aus mehr als einem Wort, wird das letzte Wort entfernt. Aus Zugriffe_Zeitraum wird Zugriffe. Damit gibt es eine direkte Übereinstimmung zur zweiten Zeile in Listing 4.4. Die Software entnimmt der Datenstruktur den Suchbegriff `http_response_code: 200` und startet eine Abfrage an die Graylog-API.

Der Zeitraum wird in ähnlicher Form ermittelt. Hierbei wird die Anzahl der zu erfassenden Wörter auf die Anzahl 2 festgelegt. Das zweite Wort entspricht ist die Zeiteinheit (Minute, Stunde, etc.), das erste Wort ist die Anzahl. Die Texterkennung der Anzahl akzeptiert sowohl Zahlen als auch Zahlwörter („ein“, „eine“, „letzte“). Wurde eine Übereinstimmung bei der Erkennung beider Werte erreicht, wird die Anzahl der Sekunden berechnet (zwei Tage entsprechen $60\ \text{s} \times 60 \times 24 \times 2 = 172800\ \text{s}$) und der Anfrage an die Graylog-API angehängt.

Listing 4.5: Anfrage der Software an Graylog mit den ermittelten Daten

```
1  GET http://10.10.12.1:9000/api/views/search/messages
2
3  {
4  "streams": [
5  "000000000000000000000001"
6  ],
7  "timerange": [
8  "relative",
9  {
10 "range": "172800"
11 }
12 ],
13 "query_string": { "type":"elasticsearch", "query_string":"
       http_response_code: 200" }
14 }
```

Der Benutzer erhält eine spezifische Fehlermeldung für den Fall, dass

- die Produktkategorie nicht ermittelt werden konnte;
- die Produktkategorie ermittelt werden konnte, aber die Eigenschaft nicht;
- der Zeitraum nicht ermittelt werden konnte.

Die Antwort der Graylog-API erfolgt in einem JSON-Objekt, welches sämtliche auf den Suchbegriff und Zeitraum passende Ereignisse beinhaltet. Die Software bestimmt die Anzahl der Ereignisse und gibt diese an den Benutzer zurück. Im Anschluss wird die nächste Nachricht verarbeitet.

Implementierung von Aliasdefinitionen

Um die Erkennung von Begriffen und die Erweiterbarkeit des Zuordnungssystems in der Datei ‚config.toml‘ zu verbessern, wurden Alias-Bezeichnungen für Produktkategorien und Eigenschaften implementiert.

- Ein Wert `"::ALIAS::"` direkt gefolgt von dem Namen einer anderen Eigenschaft der Produktkategorie stellt eine Verknüpfung von zwei Eigenschaften her. Mit der dritten Zeile von Listing 4.6 werden Anfragen für die Eigenschaft Besucher so behandelt, als wäre die Eigenschaft Besucher Zugriffe.
- Eine Eigenschaft `"::ALIAS::"` einer Produktkategorie mit dem Namen einer anderen Produktkategorie als Wert stellt eine Verknüpfung zweier Produktkategorien her. Mit der sechsten Zeile von Listing 4.6 werden Anfragen für die Produktkategorie Eins so behandelt, als wäre die Produktkategorie Webserver. Folglich können alle Eigenschaften von Webserver für die Produktkategorie Eins abgefragt werden.

Im folgenden Listing führt die Eigenschaft Besucher der Produktkategorie Eins zum Suchbegriff `http_response_code: 200`.

Listing 4.6: Beispiel für Aliasdefinitionen in der Datei config.toml

```
1 [Webserver]
2 "Besucher Zugriffe" = "http_response_code: 200"
3 "Besucher" = "::ALIAS::Besucher Zugriffe"
4
5 [Eins]
6 "::ALIAS::" = "Webserver"
```

5. Praxiseinsatz

Dieses Kapitel beschreibt den praktischen Einsatz der Software und beleuchtet zudem einen bisher nicht diskutierten Aspekt der IT-Sicherheit.

5.1. Anwendung

Der Code der entwickelten Software wurde über ein GitHub-Repository[1] veröffentlicht, sodass die Installation und Konfiguration des Bots für eine eigene Installation von Graylog wiederverwendet oder optimiert werden kann. Für den Betrieb ist es notwendig, zuvor die angeschlossenen Dienste inkl. der API-Zugriffsdaten zu konfigurieren. Im Anschluss kann die Software mit einem Python-Interpreter ausgeführt werden und ist einsatzbereit, sobald die Vorbereitungsphase abgeschlossen ist. Mithilfe der aktivierten ausführlichen Ausgaben zum Programmablauf können die Vorgänge und deren Auslöser gut zurückverfolgt werden. Im Folgenden zeigen zwei Abbildungen die Interaktion mit dem Bot über den Telegram Messenger:

Die Abbildung 5.1 zeigt, wie der Bot eine Anfrage beantwortet. Zuerst wird dem Benutzer eine Rückmeldung gesendet, sobald die Transkription fertiggestellt ist. Im Anschluss erfolgt die Beantwortung der eingesendeten Anfrage. Alle Nachrichten des Bots werden in einer kombinierten Sprachnachricht gesendet, welche zusätzlich den gesprochenen Text in geschriebener Form enthält. Zusätzlich ist es möglich, Anfragen per Textnachricht einzusenden, wie Abbildung 5.2 zeigt. Dort wurde dem Bot eine Nachricht zugestellt, welche nicht die erforderlichen Informationen beinhaltet. Der Bot prüft dies anhand der konfigurierten Schlüsselwörter und weist den Anwender auf jede fehlende Information hin.

5.2. Qualitätskontrolle

Nach Abschluss der Beschreibung zur Implementierung soll die Benutzbarkeit im geplanten Anwendungsfall kurz erläutert werden. Alle beobachteten Fälle von fehlerhaftem Verhalten der Software konnten auf Fehler bei der Transkription zurückgeführt werden. Mit dem Vergleich verschiedener Dienste (vgl. zweites Unterkapitel des Abschnitts 3.3.3) konnte die Genauigkeit der Spracherkennung bereits vor der Implementierung optimiert werden.

Durch die Möglichkeit, Anfragen auch als Textnachricht an die Software zu übermitteln, kann der Anwender im Fehlerfall auf einen alternativen Übermittlungsweg ausweichen.

Weiterhin ist es möglich, lokale Transkriptionsdienste, beispielsweise von der Tastatur-App auf einem Mobiltelefon zu nutzen. Ein zusätzliches Werkzeug für die Verbesserung der Spracherken-

[1] `https://github.com/Jomibe/ba`

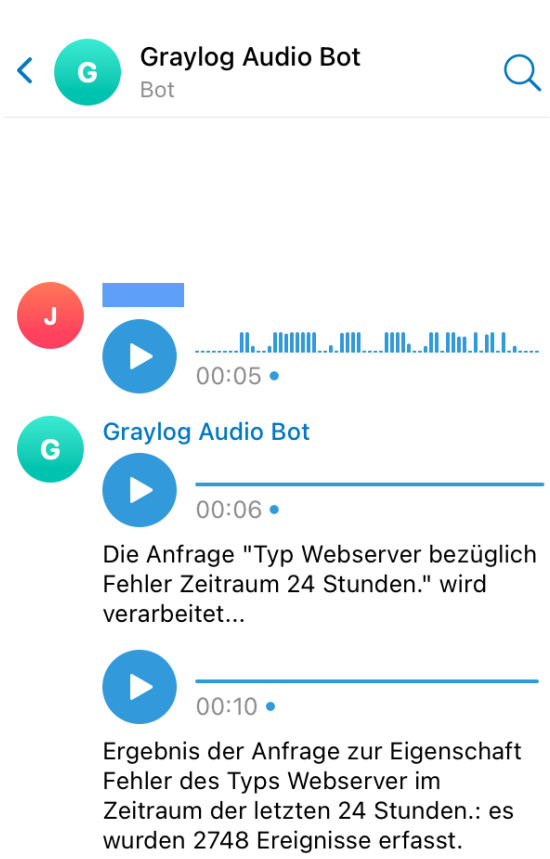

Abbildung 5.1.: Bot beantwortet eine Anfrage in Telegram.

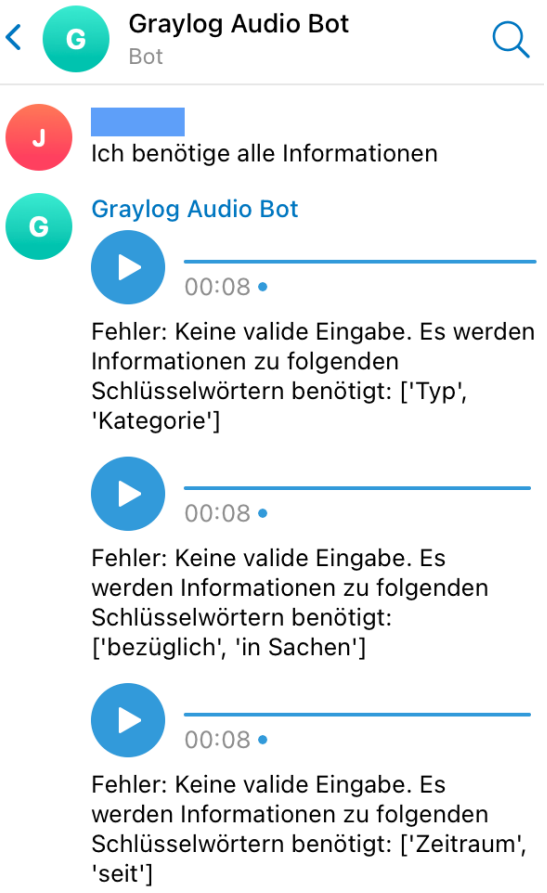

Abbildung 5.2.: Bot reagiert auf fehlerhafte Anfrage in Telegram.

nung sind die Aliasdefinitionen. Der Vergleich der verschiedenen Dienste hat bereits bei einer kleinen Stichprobe gezeigt, dass die Spracherkennung bei Fachbegriffen nicht verlässlich genug arbeitet (vgl. Tabelle 3.1). Die Definition von Begriffen aus der Alltagssprache als Alias für Fachbegriffe führt daher zu einer Verbesserung der Erkennungsquote und somit zu einer besseren Benutzbarkeit aus Sicht der Anwender. Weiterhin ist es denkbar, die Treffergenauigkeit der Transkription von Fachbegriffen mit Aliasdefinitionen indirekt zu verbessern. Als Beispiel: das Wort ‚Firewall' wird vom Transkriptionsdienst häufig zu ‚feier wohl' transkribiert (vgl. Abbildung 3.1). Indem beide Begriffe durch eine Aliasdefinition miteinander verbunden werden, wird die Anfrage trotz ursprünglicher Fehlerkennung korrekt erfasst.

5.3. IT-Sicherheit

Aus Sicht der IT-Sicherheit bringt die Verwendung des Telegram-Bots Vorteile gegenüber der Verwendung der Graylog-Webschnittstelle, sofern der Zugriff von außerhalb des Netzwerks erfolgt. Die Graylog-Instanz sammelt die Systemprotokolle sämtlicher Geräte im Netzwerk und ist damit ein besonders schützenswertes System in einem Firmennetzwerk. Der Zugriff von außerhalb sollte gut abgesichert werden. Es gibt mehrere Möglichkeiten, um webbasierte Systeme in einem internen Netzwerk über das Internet erreichbar zu machen.

Eine simple Möglichkeit besteht darin, auf dem Internetrouter eine Portweiterleitung einzurichten. Besser ist jedoch eine Veröffentlichung über einen Reverse Proxy. Ein Proxy ist ein System, welches stellvertretend für ein weiteres System eine Kommunikation mit dem gewünschten Partner übernimmt. Häufig werden Proxys in Firmennetzwerken für die Filterung von besuchten Internetseiten eingesetzt. Dabei wendet sich ein PC im internen Netzwerk an den Proxyserver und fragt eine Internetressource an. Der Proxy agiert als Vermittler und bezieht die Internetressource, bevor er diese an den PC weiterleitet. Bei einem Reverse Proxy ist das Prinzip umgekehrt. Hier übernimmt der Vermittler die Kommunikation von Geräten außerhalb eines Netzwerks (beispielsweise dem Internet) stellvertretend für die Quelle und kontaktiert den gewünschten Verbindungspartner im internen Netzwerk. Der Vorteil eines Reverse Proxys gegenüber einer Portweiterleitung besteht darin, dass die HTTP-Anfragen detailliert bearbeitet, gefiltert und protokolliert werden können. Bestimmte Adressmuster, welche für Angriffe verwendet werden, können so beispielsweise direkt blockiert werden. Der alleinige Einsatz eines Reverse Proxys mit Filter für die Veröffentlichung über das Internet reicht bei einem schützenswerten System nicht aus. Daher wird häufig eine HTTP-Authentifizierung zwischengeschaltet, welche durch die Anforderung von vom Ziel unabhängigen Zugangsdaten eine zusätzliche Sicherheitsebene schafft. Ein Reverse Proxy mit auf die Zielapplikation abgestimmten Filtermechanismen sowie einer zusätzlichen Authentifizierungsschicht bietet ausreichenden Schutz für eine Veröffentlichung im Internet.

Der Einsatz eines solchen Reverse Proxys mit Graylog ist jedoch nicht möglich, da Graylog nicht kompatibel zu Anfragen mit HTTP-Authentifizierung ist[2]. Eine Alternative zum Reverse Proxy bietet ein Dienst, welcher die Funktion eines Proxys oder Stellvertreters für die Verbindung zur Graylog-Webschnittstelle schafft. Dieser Dienst kann für beliebige Funktionen implementiert werden, welche über die Graylog REST-API Daten beziehen und weiterverarbeiten. Die in dieser Abschlussarbeit implementierte Software ist ein Beispiel für einen solchen Vermittler auf Applikationsebene [7, S. 12 ff.].

[2] Fehlerbericht und Erweiterungsvorschlag eines Anwenders, welcher Graylog über einen Reverse Proxy mit HTTP-Authentifizierung erreichbar machen möchte: `https://github.com/Graylog2/graylog2-server/issues/6831`

6. Fazit

6.1. Zusammenfassung

Das Ziel der Abschlussarbeit war die Implementierung einer Software, mit welcher Systemadministratoren sich in Echtzeit über den Status eines administrierten Netzwerkes informieren können. Die Informationsquelle für Abfragen ist die Software Graylog Open, welche Systemprotokolle sämtlicher Geräte in einem Netzwerk zentral sammelt und durchsuchbar macht. Den Administratoren sollte es ermöglicht werden, über Sprachnachrichten Anfragen zu verfassen. Die Aufgabe der zu implementierenden Software ist es, die Informationen aus der über den Messenger Telegram gesendeten Sprachnachricht zu extrahieren und damit eine Anfrage an Graylog zu stellen. Schließlich sollte die Software die Antwort von Graylog wiederum als Sprachnachricht an den Benutzer zurücksenden.

Die Implementierung der Software verlief planmäßig, sodass die gestellten Anforderungen erfüllt werden konnten. Während der Entwicklung und bei der Prüfung erster Prototypen fiel auf, dass die nicht-funktionalen Anforderungen Verarbeitungsgeschwindigkeit und Erkennungsgenauigkeit der Spracherkennung an ein Produkt, welches ausschließlich per Sprache bedient werden soll, hohe Ansprüche an die Qualität der Implementierung stellen. Durch die Verwendung von asynchronen Funktionen konnte die Verarbeitungsgeschwindigkeit erheblich reduziert werden. Zur Verbesserung der Erkennungsgenauigkeit trugen die implementierten Aliasdefinitionen bei. Alltägliche Sprache erzielt eine höhere Treffergenauigkeit bei der Transkription durch AWS als Fachsprache. Aliasdefinitionen ermöglichen es, Begriffe der Alltagssprache mit Fachbegriffen zu verknüpfen und so die niedrige Treffergenauigkeit der Fachsprache zu kompensieren.

6.2. Ausblick

Es sind einige Erweiterungen der Funktionalität der entwickelten Software denkbar.

6.2.1. Nebenläufigkeit

Die ersten Voraussetzungen für eine vollständige Nebenläufigkeit von Prozessen wurden bereits mit der Verwendung des Amazon Transcribe SDK erfüllt. Derzeit verarbeitet der Bot die eingehenden Nachrichten synchron, bzw. nacheinander. Es ist jederzeit möglich, neue Nachrichten einzusenden, diese werden jedoch sequenziell statt parallel verarbeitet.

Um die parallele Verarbeitung von eingehenden Nachrichten zu ermöglichen, ist es notwendig, asynchrone Funktionen auch in weiteren Programmmodulen einzusetzen. Weiterhin ist es denkbar, die Software für den Betrieb mit mehreren Threads anzupassen. Anfragen würden dann von einer

Softwarekomponente angenommen und von einem jeweils dafür erzeugten Thread bearbeitet werden, welche das Ergebnis asynchron an die Hauptinstanz zurückführen. Hier muss sichergestellt werden, dass pro Benutzer die Fragen in der chronologischen Reihenfolge, in welcher diese gestellt wurden, beantwortet werden. Abhilfe dazu schafft eine Gestaltung der Antwortnachrichten, welche die eingereichte Fragen nochmals wiederholt („Zu Ihrer Anfrage [...] wurden folgende Daten ermittelt [...]“) wie in Abbildung 5.1 dargestellt.

6.2.2. Datenaufbereitung

Derzeit gibt die Software für jede Anfrage die Anzahl der erfassten Ereignisse zurück. Diese Angabe ist nicht für jede Abfrage sinnvoll. Zukünftig könnten weitere Funktionen für die Aufbereitung der Informationen inkl. weiteren Schlüsselwörtern implementiert werden, welche beispielsweise den Minimal- oder Maximalwert oder in einer Umgebung mit Webservern die fünf häufigsten Statuscodes zurückgeben.

6.2.3. Verwaltung per Telegram

Änderungen am Verhalten der Software und insbesondere an den Zuordnungen der Suchbegriffe zu den Schlüsselwörtern können derzeit nur über eine Anpassung von Konfigurationsdateien vorgenommen werden. Es könnte ein Rollenkonzept eingeführt werden, welches es bestimmten Benutzern erlaubt, Änderungen an den Zuordnungen vorzunehmen und neue Abfragen zu definieren.

A. Anhang

A.1. Konsolenausgabe beim Start des Programms

```
Info: Detaillierte Ausgaben zum Programmablauf sind eingeschaltet.
Info: Starte Vorbereitungen...
Info: Bereite die Protokollaufzeichnung in eine Datei vor
Erfolg: Name für die Protolldatei: log/**entfernt**.log
Info: TELEGRAM_BOT_TOKEN = **entfernt**
Info: GRAYLOG_USERNAME = **entfernt**
Info: GRAYLOG_PASSWORD = **entfernt**
Info: AWS_ACCESS_KEY_ID = **entfernt**
Info: AWS_SECRET_ACCESS_KEY = **entfernt**
Info: AWS_S3_BUCKET_NAME = **entfernt**
Info: AWS_S3_BUCKET_VOICE_DIR = voice/
Info: AWS_REGION = eu-central-1
Info: Prüfung der Verbindung zur Telegram API...
Erfolg: Telegram API ist in 172.857ms erreichbar
Info: Prüfung der Verbindung zur Graylog API...
Info: Abrufen eines weiteren temporären Authentifizierungsschlüssels für
    Graylog...
Erfolg: Authentifizierungsschlüssel **entfernt** erfolgreich bezogen
Erfolg: Graylog API ist erreichbar
Info: Stelle Verbindung zu AWS Transcribe in der Region eu-central-1 her
Info: Prüfe Verbindung zu AWS Polly in der Region eu-central-1
Erfolg: Verbindung zu AWS Polly aufgebaut
Info: Wiederherstellen der aktuellen update_id...
Erfolg: Wiederherstellung der aktuellen update_id **entfernt**
   abgeschlossen.
Info: Importiere Inhalt der Datei config.toml
Erfolg: Import erfolgreich
Info: Prüfe auf Aliase
Info: Prüfe auf Alias bei Wert http_response_code: 200 von Eigenschaft
   Besucher Zugriffe
Info: Prüfe auf Alias bei Wert ::ALIAS::Besucher Zugriffe von
   Eigenschaft Besucher
Erfolg: Eigenschafts-Alias Besucher mit Ziel Besucher Zugriffe erkannt
Info: Verbinde Ziel Besucher Zugriffe mit Quelle Besucher
Info: Prüfe auf Alias bei Wert http_response_code:[500 TO 599] von
   Eigenschaft Interne Fehler
Info: Prüfe auf Alias bei Wert ::ALIAS::Besucher von Eigenschaft Fehler
Erfolg: Eigenschafts-Alias Fehler mit Ziel Besucher erkannt
Info: Verbinde Ziel Besucher mit Quelle Fehler
Info: Prüfe auf Alias bei Wert ::ALIAS::Fehler von Eigenschaft drei
```

```
Erfolg: Eigenschafts-Alias drei mit Ziel Fehler erkannt
Info: Verbinde Ziel Fehler mit Quelle drei
Info: Prüfe auf Alias bei Wert http_response_code:[400 TO 499] von
   Eigenschaft Erreichbarkeit
Info: Prüfe auf Alias bei Wert ::ALIAS::Erreichbarkeit von Eigenschaft
   Verfügbarkeit
Erfolg: Eigenschafts-Alias Verfügbarkeit mit Ziel Erreichbarkeit erkannt
Info: Verbinde Ziel Erreichbarkeit mit Quelle Verfügbarkeit
Info: Prüfe auf Alias bei Wert ::ALIAS::fuenf von Eigenschaft vier
Erfolg: Eigenschafts-Alias vier mit Ziel fuenf erkannt
Info: Verbinde Ziel fuenf mit Quelle vier
Info: Prüfe auf Alias bei Wert ::ALIAS::vier von Eigenschaft fuenf
Erfolg: Eigenschafts-Alias fuenf mit Ziel vier erkannt
Info: Verbinde Ziel vier mit Quelle fuenf
Erfolg: Typ-Alias Eins mit Ziel Webserver erkannt
Info: Stelle Verbindung zum Zieltyp Webserver her
Info: Der Typ Eins -> Webserver wurde verknüpft
Info: Prüfe auf Alias bei Wert http_response_code: 200 von Eigenschaft
   Besucher Zugriffe
Info: Prüfe auf Alias bei Wert http_response_code: 200 von Eigenschaft
   Besucher
Info: Prüfe auf Alias bei Wert http_response_code:[500 TO 599] von
   Eigenschaft Interne Fehler
Info: Prüfe auf Alias bei Wert http_response_code: 200 von Eigenschaft
   Fehler
Info: Prüfe auf Alias bei Wert http_response_code: 200 von Eigenschaft
   drei
Info: Prüfe auf Alias bei Wert http_response_code:[400 TO 499] von
   Eigenschaft Erreichbarkeit
Info: Prüfe auf Alias bei Wert http_response_code:[400 TO 499] von
   Eigenschaft Verfügbarkeit
Info: Prüfe auf Alias bei Wert ::ALIAS::vier von Eigenschaft vier
Erfolg: Eigenschafts-Alias vier mit Ziel vier erkannt
Info: Verbinde Ziel vier mit Quelle vier
Info: Prüfe auf Alias bei Wert ::ALIAS::vier von Eigenschaft fuenf
Erfolg: Eigenschafts-Alias fuenf mit Ziel vier erkannt
Info: Verbinde Ziel vier mit Quelle fuenf
Info: Prüfe auf Alias bei Wert INBOX von Eigenschaft Empfangenene Mails
Info: Prüfe auf Alias bei Wert "NOQUEUE: reject" von Eigenschaft
   Blockierte Mails
Erfolg: Typ-Alias Zwei mit Ziel Mailserver erkannt
Info: Stelle Verbindung zum Zieltyp Mailserver her
Info: Der Typ Zwei -> Mailserver wurde verknüpft
Info: Prüfe auf Alias bei Wert INBOX von Eigenschaft Empfangenene Mails
Info: Prüfe auf Alias bei Wert "NOQUEUE: reject" von Eigenschaft
   Blockierte Mails
Info: Prüfe auf Alias bei Wert "[UFW BLOCK]" von Eigenschaft Blockierte
   Pakete
Info: Prüfe auf Alias bei Wert ::ALIAS::Blockierte Pakete von
   Eigenschaft Blockiert
Erfolg: Eigenschafts-Alias Blockiert mit Ziel Blockierte Pakete erkannt
```

```
Info: Verbinde Ziel Blockierte Pakete mit Quelle Blockiert
Info: Prüfe auf Alias bei Wert ::ALIAS::Blockiert von Eigenschaft
   Abgewiesene Pakete
Erfolg: Eigenschafts-Alias Abgewiesene Pakete mit Ziel Blockiert erkannt
Info: Verbinde Ziel Blockiert mit Quelle Abgewiesene Pakete
Info: Prüfe auf Alias bei Wert  von Eigenschaft Top Port
Info: Prüfe maximale Wortanzahl von Typ- und Eigenschaftsbezeichnungen
   der Konfigurationsdatei
Info: Prüfe Typ Webserver
Erfolg: Typ Webserver erhöht die maximale Wortanzahl von
   Typbezeichnungen auf 1
Info: Prüfe Eigenschaft Besucher Zugriffe
Erfolg: Eigenschaft Besucher Zugriffe erhöht die maximale Wortanzahl von
    Eigenschaftsbezeichnungen auf 2
Info: Prüfe Eigenschaft Besucher
Info: Prüfe Eigenschaft Interne Fehler
Info: Prüfe Eigenschaft Fehler
Info: Prüfe Eigenschaft drei
Info: Prüfe Eigenschaft Erreichbarkeit
Info: Prüfe Eigenschaft Verfügbarkeit
Info: Prüfe Eigenschaft vier
Info: Prüfe Eigenschaft fuenf
Info: Prüfe Typ Eins
Info: Prüfe Eigenschaft Besucher Zugriffe
Info: Prüfe Eigenschaft Besucher
Info: Prüfe Eigenschaft Interne Fehler
Info: Prüfe Eigenschaft Fehler
Info: Prüfe Eigenschaft drei
Info: Prüfe Eigenschaft Erreichbarkeit
Info: Prüfe Eigenschaft Verfügbarkeit
Info: Prüfe Eigenschaft vier
Info: Prüfe Eigenschaft fuenf
Info: Prüfe Typ Mailserver
Info: Prüfe Eigenschaft Empfangenene Mails
Info: Prüfe Eigenschaft Blockierte Mails
Info: Prüfe Typ Zwei
Info: Prüfe Eigenschaft Empfangenene Mails
Info: Prüfe Eigenschaft Blockierte Mails
Info: Prüfe Typ Firewall
Info: Prüfe Eigenschaft Blockierte Pakete
Info: Prüfe Eigenschaft Blockiert
Info: Prüfe Eigenschaft Abgewiesene Pakete
Info: Prüfe Eigenschaft Top Port
Erfolg: Prüfung erfolgreich. Maximale Wortanzahl Typbezeichnungen: 1 .
   Maximale Wortanzahl Eigenschaftsbezeichnungen: 2
Erfolg: Vorbereitungen erfolgreich abgeschlossen
Info: Prüfung auf neue Ereignisse der Telegram-API...
Info: Rufe Aktualisierungen von der Telegram API ab
```

A.2. Konsolenausgabe bei der Verarbeitung einer eintreffenden Nachricht

```
Info: Rufe Aktualisierungen von der Telegram API ab
Erfolg: Antwort in 758.149ms erhalten
Erfolg: Es liegen neue Ereignisse vor
Info: Name : **entfernt**
Info: Chat_id : **entfernt**
Info: Aktuelle update_id **entfernt** speichern...
Erfolg: Speichern der aktuellen update_id **entfernt** abgeschlossen.
Info: Die aktuelle update_id ist nun **entfernt**
Info: Sprachnachricht erhalten
Info: file_id : **entfernt**
Info: Dateityp : audio/ogg
Info: Ermittlung des Dateipfads für den Download über die Telegram API
Erfolg: Dateipfad bezogen
Info: Lade Datei herunter. URL: https://api.telegram.org/file/bot**
   entfernt**/voice/file_**entfernt**.oga
Info: Speichere auf dem Dateisystem. Dateipfad: files/**entfernt**.oga
Info: Speichern abgeschlossen
Info: Beginne mit Echtzeitübersetzung Sprache zu Text
Info: Beginne mit Datenübertragung zu AWS Transcribe
Info: Bereite Verarbeitung der von AWS Transcribe übermittelten Daten
   vor
Info: Übertrage Abschnitt von 16384B der Datei files/**entfernt**.oga an
    AWS Transcribe
Erfolg: Datenübertragung abgeschlossen. Warte auf Abschluss des
   Transkriptionsprozesses
Info: Transkription wird durchgeführt, nicht finales Ergebnis wurde ü
   bermittelt: Typ.
Info: Transkription wird durchgeführt, nicht finales Ergebnis wurde ü
   bermittelt: Typ eins
Info: Transkription wird durchgeführt, nicht finales Ergebnis wurde ü
   bermittelt: Typ eins Bezug.
Info: Transkription wird durchgeführt, nicht finales Ergebnis wurde ü
   bermittelt: Typ eins bezüglich drei
Info: Transkription wird durchgeführt, nicht finales Ergebnis wurde ü
   bermittelt: Typ eins bezüglich drei Zeitraum.
Info: Transkription wird durchgeführt, nicht finales Ergebnis wurde ü
   bermittelt: Typ eins bezüglich drei Zeitraum einen Tag.
Info: Transkription wird durchgeführt, nicht finales Ergebnis wurde ü
   bermittelt: Typ eins bezüglich drei Zeitraum einen Tag.
Erfolg: Transkription abgeschlossen. Ergebnis: Typ eins bezüglich drei
   Zeitraum einen Tag.
Info: Sende Text Die Anfrage "Typ eins bezüglich drei Zeitraum einen Tag
   ." wird verarbeitet... an Chat **entfernt** in einer kombinierten
   Sprachnachricht.
Info: Beginne mit Sprachsynthese
Erfolg: Der Sprachsynthesevorgang **entfernt** wurde gestartet
Info: Schreibe in Datei
```

```
Info: Sende Audiodatei files/**entfernt**.mp3 als Sprachnachricht
Erfolg: Nachricht in 229.919ms übermittelt
Info: Untersuche Nachricht auf Schlüsselwörter...
Info: Prüfe auf Typ
Erfolg: Schlüsselwort Typ erkannt. Folgende 1 Wörter: eins
Info: Untersuche Nachricht auf Schlüsselwörter...
Info: Prüfe auf bezüglich
Erfolg: Schlüsselwort bezüglich erkannt. Folgende 2 Wörter: drei
   Zeitraum
Info: Untersuche Nachricht auf Schlüsselwörter...
Info: Prüfe auf Zeitraum
Erfolg: Schlüsselwort Zeitraum erkannt. Folgende 2 Wörter: einen Tag.
Info: Extrahiere Informationen zum angeforderten Zeitraum aus einen Tag.
Info: Ermittle Anzahl der Zeiteinheiten aus einen
Erfolg: Anzahl der Zeiteinheiten wurde bestimmt: 1
Info: Ermittle Zeiteinheit aus Tag.
Erfolg: Zeiteinheit Tag wurde bestimmt
Info: Informationen der letzten 86400 Sekunden werden ausgewertet
Info: Prüfe eins auf Übereinstimmung mit Typ Webserver
Info: Vergleiche eins und webserver miteinander
Info: Prüfe eins auf Übereinstimmung mit Typ Eins
Info: Vergleiche eins und eins miteinander
Erfolg: Direkte Übereinstimmung
Erfolg: Typ Eins identifiziert
Info: Prüfe drei Zeitraum auf Übereinstimmung mit Eigenschaft Besucher
   Zugriffe
Info: Vergleiche drei zeitraum und besucher zugriffe miteinander
Info: Vergleiche Untermenge drei mit besucher zugriffe
Info: Prüfe drei Zeitraum auf Übereinstimmung mit Eigenschaft Besucher
Info: Vergleiche drei zeitraum und besucher miteinander
Info: Vergleiche Untermenge drei mit besucher
Info: Prüfe drei Zeitraum auf Übereinstimmung mit Eigenschaft Interne
   Fehler
Info: Vergleiche drei zeitraum und interne fehler miteinander
Info: Vergleiche Untermenge drei mit interne fehler
Info: Prüfe drei Zeitraum auf Übereinstimmung mit Eigenschaft Fehler
Info: Vergleiche drei zeitraum und fehler miteinander
Info: Vergleiche Untermenge drei mit fehler
Info: Prüfe drei Zeitraum auf Übereinstimmung mit Eigenschaft drei
Info: Vergleiche drei zeitraum und drei miteinander
Info: Vergleiche Untermenge drei mit drei
Erfolg: Übereinstimmung gefunden
Erfolg: Übereinstimmung mit drei
Info: Führe Abfrage http_response_code: 200 aus
Info: Führe Abfrage http_response_code: 200 im Zeitraum der letzten
   86400 Sekunden in Graylog aus
Erfolg: Ergebnis der Abfrage: 2732 Ereignisse
Info: Sende Text Ergebnis der Anfrage zur Eigenschaft drei des Typs Eins
    im Zeitraum der letzten einen Tag.: es wurden 2732 Ereignisse
   erfasst. an Chat **entfernt** in einer kombinierten Sprachnachricht.
Info: Beginne mit Sprachsynthese
```

```
Erfolg: Der Sprachsynthesevorgang **entfernt** wurde gestartet
Info: Schreibe in Datei
Info: Sende Audiodatei files/**entfernt**.mp3 als Sprachnachricht
Erfolg: Nachricht in 266.194ms übermittelt
Info: Prüfung auf neue Ereignisse der Telegram-API...
```

Literaturverzeichnis

[1] Roy Thomas Fielding. „Architectural Styles and the Design of Network-based Software Architectures“. 2000.

[2] Veikko Krypczyk. *Handbuch für Softwareentwickler*. ger. 2., akt. u. überarb. Aufl. Rheinwerk Computing. Bonn: Rheinwerk Verlag, 2022. ISBN: 9783836279772.

[3] Johannes Ernesti. *Python 3: Das umfassende Handbuch*. ger. 6., akt. Aufl. Rheinwerk Computing. Bonn: Rheinwerk Verlag, 2020. ISBN: 9783836279260.

[4] Jens Jacobsen. *Praxisbuch Usability und UX: Was alle wissen sollten, die Websites und Apps entwickeln*. ger. 3., akt. u. erw. Aufl. Rheinwerk Computing. Bonn: Rheinwerk Verlag, 2022. ISBN: 9783836288408.

[5] Peter Saint-Andre, Salvatore Loreto, Stefano Salsano u. a. *Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP*. RFC 6202. Apr. 2011. DOI: 10.17487/RFC6202. URL: https://www.rfc-editor.org/info/rfc6202.

[6] Jakob Nielsen. *Usability engineering*. eng. [Nachdr.]. Amsterdam [u.a: Morgan Kaufmann, 2005. ISBN: 0125184069.

[7] Bundesamt für Sicherheit in der Informationstechnik. *Sicheres Bereitstellen von Web-Angeboten (ISi-Web)*. 2008. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Internetsicherheit/isi_web_server_leitlinie.pdf (besucht am 09. 10. 2022).

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis