Fachbereich VI Informatik und Medien

-Bachelorarbeit-

im Rahmen des Studiengangs Medieninformatik Bachelor of Science

Implementierung einer Überwachung der Verarbeitungszeit von Produktinformationen in einer Microservice-Architektur

Erstprüfer: Prof. Dr.-Ing. Edzard Höfig

Zweitprüfer: Prof. Dr. Schimkat

WS 2021/2022

Steffen Scheller (Matr. Nr. 923518)

Zwinglistraße 22

10555 Berlin

- Bringmeister GmbH -

Erklärung

Selbstständigkeitserklärung

Hiermit versichere ich, die vorliegende Abschlussarbeit im Studiengang Medieninformatik (BA) der BHT Berlin, eigenständig und ausschließlich unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt zu haben. Sämtliche Stellen, die wörtlich oder sinngemäß zitiert sind, wurden deutlich gekennzeichnet und finden sich im abschließenden Literaturverzeichnis wieder.

_____

Steffen Scheller, Berlin 2022

Abstrakt

Die Grundlage dieser Bachelorarbeit ist die Implementierung eines Services zur Überwachung der Verarbeitungszeit von Produktinformationen in einer Microservice-Architektur. Der in der Programmiersprache Kotlin entwickelte Microservice ermöglicht der Firma Bringmeister GmbH, die Durchlaufzeiten des Verarbeitungsprozesses von Verfügbarkeitsdaten einzelner Produkte zu ermitteln.

Der Microservice stellt Funktion zum Herunterladen dieser Daten bereit. Anhand dieser ermittelten Daten ist die Firma Bringmeister GmbH imstande, ihren Verarbeitungsprozess in Hinblick auf die Durchlaufzeiten und Last zu analysieren.

Danksagung

Die vorliegende Abschlussarbeit entstand im Studiengang Medieninformatik (Bachelor) der Berliner Hochschule für Technik, in Zusammenarbeit mit der Bringmeister GmbH. Ziel der Abschlussarbeit ist die Implementierung einer Überwachung der Verarbeitungszeit von Produktinformationen in einer Microservice-Architektur.

Mein herzlicher Dank gilt der Firma Bringmeister und ganz besonders dem gesamten Team Connect, für die (mittlerweile) fast zweijährige Unterstützung in der Praktikumsphase, Werkstudententätigkeit und Bachelor-Arbeit.

Besonderer Dank geht an meinen Betreuer Thomas Uhrig. Er stand mir während meiner gesamten Zeit bei der Bringmeister GmbH jederzeit mit viel Engagement zur Seite.

Ebenso möchte ich mich bei meinem betreuenden Professor, Dr.-Ing. Edzard Höfig, von der Berliner Hochschule für Technik, bedanken.

Mein abschließender Dank geht an meine Familie und Freunde, die mich während des gesamten Studiums unterstützt haben. Euer Rückhalt hat mir sehr geholfen.

_____

Steffen Scheller, Berlin 2022.

Inhalt

Abkürzungsverzeichnis

API	Application Programming Interface
CI	Continuous-Integration
CD	Continuous-Delivery
CPU	Central Processing Unit
CSV	Comma-Separated-Values
ERP	Enterprise Resource Planning
HTTP	Hypertext Transfer Protocol
ISO	Internationale Organisation für Normung
JAXB	Jakarta XML Binding (früher Java Architecture for XML Binding)
MINE	Multipurpose Internet Mail Extensions
REST	Representational State Transfer
SKU	Stock Keeping Unit
SOAP	Simple Object Access Protocol
XML	Extensible Markup Language
XSD	XML-Schema Definition

1 Einleitung

Die Firma Bringmeister GmbH möchte den Kunden in ihrem Online-Shop ein optimales Einkaufserlebnis bieten. Dafür müssen dem Kunden Informationen, wie z.B. die Verfügbarkeit der Produkte, möglichst in Echtzeit dargestellt werden. Andernfalls besteht die Gefahr, dass der Kunde etwas bestellen will, was bereits ausverkauft ist.

Aus diesem Grund benötigt Bringmeister ein Verfahren, um Aussagen über den Zeitraum ihrer definierten Echtzeit treffen zu können. Um das zu bewerkstelligen, muss der Verarbeitungsprozess der Daten durch das System analysiert werden.

1.1 Zielsetzung der Arbeit

Das Ziel dieser Arbeit ist, einen Microservice zu entwickeln, der die Zeit des Verarbeitungsprozesses von Produktdaten durch die Microservice-Architektur messen und aufzeichnen kann.

Die durch den Microservice gesammelten und aufbereiteten Daten sollen über eine Schnittstelle zum Herunterladen bereitgestellt werden. Die heruntergeladenen Daten werden mittels eines externen Programms als Diagramm grafisch dargestellt und ausgewertet.

Dies soll Erkenntnisse darüber liefern, wie lange der Verarbeitungsprozess von Produktdaten tatsächlich dauert und ob ein Zusammenhang zwischen der gemessenen Zeit des Verarbeitungsprozesses und der Anzahl der aufkommenden Ereignisse (engl. Events) existiert. Daraus ergeben sich die folgenden beiden Fragen, die mit den Ergebnissen beantwortet werden sollen:

1. Wie lange dauert der durchschnittliche Verarbeitungsprozess von Produktdaten?
2. Besteht ein Zusammenhang zwischen der Durchlaufzeit und der Last des Systems?

1.2 Abgrenzung zu verwandten Themen

Im IT-Bereich lassen sich Prozesse, Betriebssysteme, Netzwerke, Software- und Webanwendungen mittels sogenannter Monitoring Systeme überwachen. Diese stellen eine Vielzahl von unterschiedlichen Informationen bereit, mit denen ein besseres Verständnis über die Abläufe der Systeme erlangt werden und bei Ausfällen schlagen diese Alarm.

Dabei gibt es eine Vielzahl an Monitoring Systemen für unterschiedliche Aufgabebereiche (z.B. DataDog oder Kibana) für ein eventbasiertes Monitoring. Diese sind für den in dieser Arbeit betrachten Anwendungsfall nicht geeignet, da diese Monitoring Systeme zwar zur

Überwachung von Systemen und Services eingesetzt werden, diese hauptsächlich nur Fehlerfälle erfassen [TenMedia, 2022].

1.3 Struktur der Arbeit

Als erstes werden die Themengebiete erläutert und beschrieben, welche im Kapitel [2. Grundlagen] darstellen werden. Damit soll ein Überblick über die eingesetzten Techniken und Methoden gegeben werden.

Danach wird in dem Kapitel [3. Konzeption] die Herangehensweise beschrieben, welche Voraussetzungen bestehen und wie die daraus resultierenden Ideen für die Umsetzung angewandt werden.

Im nächsten Kapitel [4. Implementierung] wird aufgezeigt, welche Mechanismen und Verfahren tatsächlich genutzt und umgesetzt worden sind.

In dem Kapitel [5. Auswertung] wird aufgezeigt, welche Ergebnisse erlangt werden konnten und wie diese bewertet worden sind. Dabei wird auf die Schwierigkeiten eingegangen, die entstanden sind, wie damit umgegangen worden ist und was daraus abgeleitet werden konnte.

Im letzten Kapitel [6. Abschließende Betrachtung] kommt das Fazit, in dem die gesammelten und erhaltenen Erkenntnisse zusammengefasst werden, welche die Grundlage zur Bewertung der Ergebnisse dieser Arbeit darstellen. Des Weiteren werden die Grenzen und Kritik angesprochen und zum Schluss wird noch ein kurzer Ausblick über eventuelle Erweiterungsmöglichkeiten gegeben.

2 Grundlagen

In diesem Kapitel werden die relevanten Grundlagen dieser Arbeit aufgeführt und deren Bedeutungen beschrieben.

2.1 Microservice

Ein Microservice ist ein Architekturstil [Richardson, 2014] das weitgehend der Unix-Philosophie „Write programs that do one thing and do it well“ [Raymond, 2003, S. 34] entspricht. Es lassen sich dadurch große Softwareanwendungen in kleine, eigenständige und unabhängige Dienste aufteilen. Diese sind im Idealfall in fachliche Domänen unterteilt und stellen damit die einzelnen Funktionalitäten der Software breit. Die Kommunikation der Dienste kann über REST-APIs realisiert werden. Dadurch, dass die Dienste voneinander entkoppelt sind, ist die Wartbarkeit wesentlich leichter und es ist einfacher eine Continuous-Integration/Continuous-Delivery (CI/CD) Pipeline aufzubauen. Hierdurch lassen sich Änderungen schneller produktiv schalten.

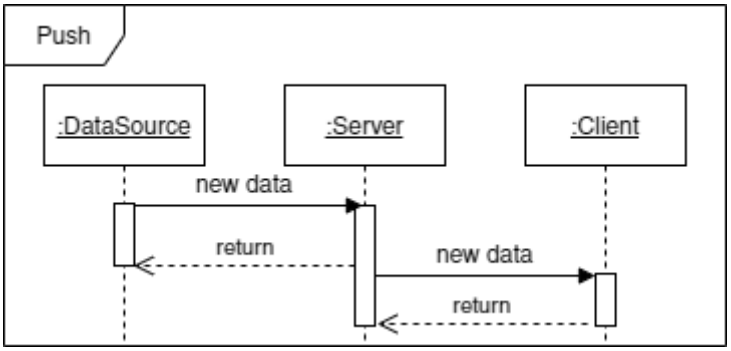
Dem gegenüber stehen Makro-Architekturen oder auch monolithische Architekturen, die auch als Monolithen bezeichnet werden. Mit einem Monolith ist gemeint, dass eine Domaine, wie z.B. die Backend-Architektur, in einem einzigen Service umgesetzt wird. Es kann bei dieser Architektur ein Nachteil sein, wenn alles in einem Service implementiert ist, da Änderungen an einer Stelle, Auswirkungen auf eine oder mehrere Stellen hervorrufen oder auslösen können. Dadurch ist die Wartbarkeit einer solchen Architektur oftmals schwerfällig und langsam.

2.2 Push-, Pull- und Polling-Verfahren

Die Kommunikation zwischen den einzelnen Services kann auf verschiedenen Wegen oder mittels verschiedener Strategien realisiert werden. Diese arbeiten mit unterschiedlichen Ansätzen. In der nachfolgenden Grafik sind die drei Verfahren Push, Pull und Polling schematisch dargestellt.

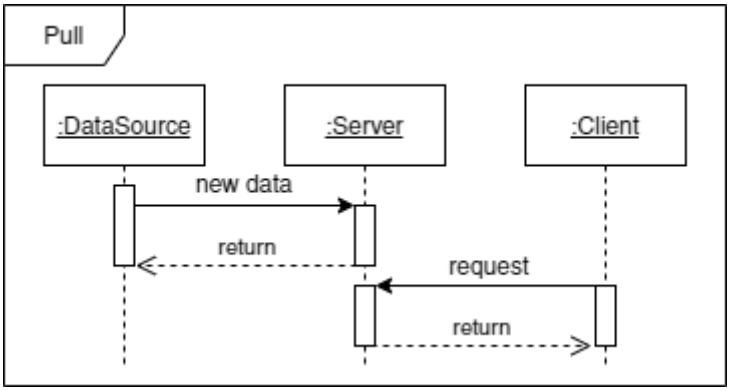
Die Begriffe Datenkonsument (Client) und Datenlieferant (Server) werden im weiteren Verlauf diese beiden Arten verwendet.

2.2.1 Push-Verfahren



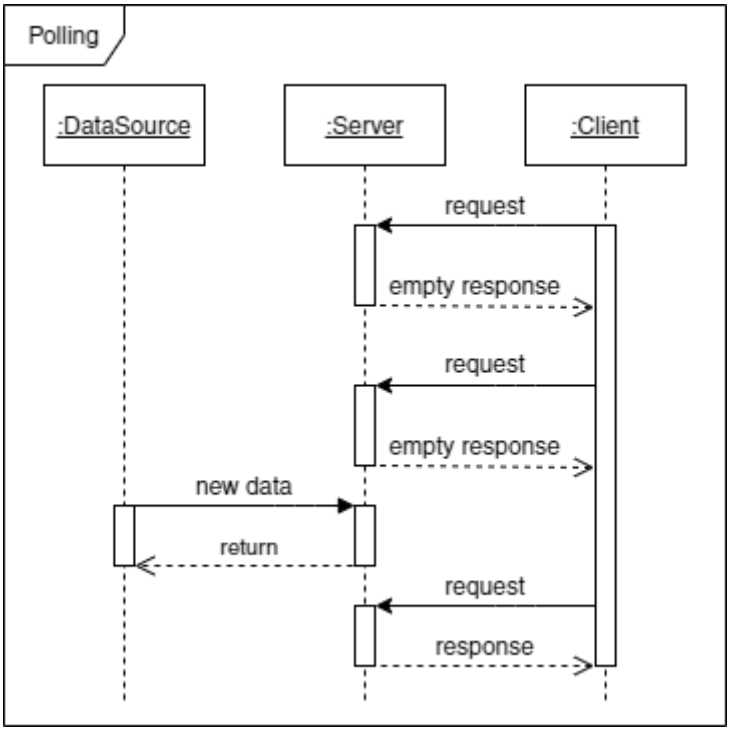
Beim Push-Verfahren werden die Daten vom Datenlieferant an seinen Datenkonsument geschickt, sobald dem Datenlieferanten die neuen Daten von der DataSource zur Verfügung stehen. Dafür benötigt der Server aber eine Auswertelogik, mit der Änderungen der Werte erkannt werden und eine Liste all seiner Konsumenten, die er mit diesen Daten benachrichtigen soll [Buschmann, 1996, S. 14].

2.2.2 Pull-Verfahren



Beim Pull-Verfahren muss der Datenkonsument sich die Daten beim Datenlieferanten selbst abholen, wenn er diese benötigt, da der Server den Client nur darüber in Kenntnis setzt, sich etwas geändert hat. Bei diesem Verfahren muss beachtet werden, dass der Client zwar darüber informiert, dass der Server neue Daten von der DataSource erhalten hat aber dabei nicht mitteilt, was sich geändert hat [Buschmann, 1996, S. 14].

2.2.3 Polling-Verfahren



Beim Polling-Verfahren fragt der Datenkonsument beim Datenlieferant immer wieder nach, ob sich etwas geändert hat. Wenn die dies der Fall ist, da der Server von der DataSource neue Daten erhalten hat, bekommt der Client die Daten vom Server bei der nächsten Abfrage [Schnell & Hoyer, 1986, S. 2].

Dieses Verfahren hat folgenden Nachteil: Es wird nie wirklich Echtzeit sein, da die Abfragen immer in einem Intervall durchgeführt werden. Deses Intervall hat immer einen Versatz (eine Verzögerung) zwischen den einzelnen Abfragen. Setzt man den Versatz des Intervalls auf die kleinstmögliche Einheit, welche im Bereich von Nanosekunden liegen kann, wird vermehrt Last (s. Kapitel [2.3]) erzeugt. Dadurch kann die Performance des Systems teils erhebliche Einschränkungen erfahren.

2.3 Last

Mit Last wird der Arbeitsaufwand bezeichnet, dem ein System ausgesetzt ist und mit welcher es umgehen muss. Die Lasten, die in einem System auftreten, können unterschiedlicher Art sein. Es können Anfragen an das System sein, parallellaufende Prozesse oder aber auch aktive Benutzer.

Bei Messungen und Tests kann Last eines Systems ein deutlicher Effekt sein, der die Messdaten beeinflusst. So kann die Antwortzeit bei einem erhöhten Aufkommen von

Anfragen an das System nur dann genau bestimmt werden, wenn die Voraussetzungen einer Messung (wie z.B. die Anzahl der Anfragen und aktive Prozesse) präzise definiert sind.

2.4 Asynchrone Verarbeitung und Synchronisation

In diesem Kapitel wird erläutert, was die Unterschiede zwischen Synchron und Asynchron sind und wie sichergestellt werden kann, dass es zu keiner Verfälschung der Daten kommt.

2.4.1 Asynchrone Verarbeitung

Bei der asynchronen Verarbeitung spricht man von Parallelität, also Gleichzeitigkeit. Dabei wird eine komplexe Aufgabe in mehrere Teile aufgeteilt, welche dann parallel abgearbeitet werden. Durch diese Aufteilung wird die Verarbeitungszeit der komplexen Aufgabe verkürzt. Dies ist gegeben, wenn eine Anwendung mehrere Prozessoren zur Verfügung hat, auf denen die einzelnen Prozesse verteilt und so zum gleichen Zeitpunkt verarbeitet werden können. Bei einer asynchronen Verarbeitung muss auf die Synchronisation der Daten geachtet werden, da diese sonst möglicherweise verfälscht werden.

2.4.2 Synchronisation

Mit Synchronisation ist der zeitliche Abgleich von Prozessen bezeichnet. Dies ist von essenzieller Bedeutung, weil damit sichergestellt wird, dass parallellaufende Prozesse an einem definierten Punkt zusammengeführt werden und damit wieder synchron sind. Dadurch behalten die Daten bei der parallelen Verarbeitung (s. Kapitel [2.4.1]) ihren konsistenten Zustand. Wenn dies nicht beachtet wird, kann es zu Asynchronität kommen, wodurch die Prozesse nicht mehr zeitlich abgeglichen oder koordiniert sind.

2.5 Zeitdaten

Der Grundaspekt dieser Arbeit ist der Umgang mit Zeitdaten (z.B. Zeitformate und Durchlaufzeiten), sowie deren Messung und Berechnung. Anschließend werden diese Begriffe aufgeführt und mittels ihrer Definition eingegrenzt. Weiterhin wird beschrieben welche Messstrukturen und Messgrenzen es gibt.

2.5.1 Durchlaufzeit

Die Durchlaufzeit beschreibt eine Zeitspanne, die „zwischen dem Beginn des ersten und dem Abschluss des letzten Arbeitsgangs verstreicht. Die Durchlaufzeit eines Auftrages ist

definiert als die Summe der Bearbeitungs-, Transport- und Wartezeiten auf allen Stufen der Produktion“ [Voigt, 2022] .

2.5.2 Zeitintervall

Ein Zeitintervall ist die Dauer, die den Abstand zwischen zwei Punkten auf einer Zeitskala darstellt. Der zeitliche Abstand, wenn er sich im selben Bezugssystem befindet, wird als Zeitintervall bezeichnet, dieses kann als Delta t (**Δt**) dargestellt werden.

2.5.3 Zeitformate

Die unterschiedlichen Datums- und Zeit Formate, die verwendet werden können, beziehen sich auf eine Unterscheidung von zwei Hauptansichten der Zeitachse, die als Mensch- und Maschinezeit verstanden werden [Ullenboom, 2021, Kapitel 15.8]. Die Menschenzeit ist in Einheiten wie Tage und Minuten aufgelöst. Bei der Maschinenzeit wird die Zeit im Nanosekundenbereich aufgelöst. Der Startpunkt der Maschinenzeit wird als Epoche bezeichnet, die bei UNIX am 01.01.1970 beginnt, was den hexadezimal: 00 00 00 00 entspricht. Diese Zeitformate sind unveränderliche Formate, die bei einer Manipulation eine Kopie dieser erstellen und das Original nicht verändern.

Zwei der meistverwendeten Formate sind die folgenden beiden. Diese lassen sich jeweils in drei eigene Darstellungsformen unterteilen.

LocalDateTime:

- LocalDate
- LocalTime
- LocalDateTime

OffsetDateTime:

- OffsetDate
- OffsetTime
- OffsetDateTime

Mit diesen beiden Formaten können alle Datums- und Uhrzeitangaben mit einer Genauigkeit von Nanosekunden gespeichert werden. Der Unterschied zwischen den beiden genannten Formaten liegt darin, dass das OffsetDateTime Format keine Zeitzone, sondern ein Offset (Versatz) speichert, mit dem immer der gleiche Zeitpunkt mit einem lokalen Versatz dargestellt wird. Aus diesem Grund sollte das OffsetDateTime Format zur Speicherung von Zeitangaben in Datenbanken genutzt werden, da bei den lokalen

Zeitformaten die Sommer -und Winterzeit berücksichtig werden muss, da es sonst zu Fehlern oder leeren Datensätzen kommen kann [Srivastava, 2020].

2.6 Korrelation

Wie in Kapitel [1.1] dargestellt ist ein Ziel dieser Arbeit die Ermittlung eines Zusammenhanges zwischen der gemessenen Zeit des Verarbeitungsprozesses und der Anzahl der aufkommenden Ereignisse. Um die in Kapitel [5] dargestellten Messergebnisse besser vergleichen zu können, wird der Korrelationskoeffizient nach Pearson herangezogen [Brückler, 2018, S. 117 ff.].

Mit der Korrelation wird der Zusammenhang zwischen zwei oder mehreren Variablen bezeichnet, der sich im positiven oder im negativen Bereich befinden kann. Dabei wird die Korrelation mit Hilfe eines Korrelationskoeffizienten ausgedrückt. Dieser gibt anhand des Betrags (Stärke) und des Vorzeichens (Richtung) den Zusammenhang an [Humboldt-Universität, 2019].

Der Zusammenhang der Richtung ist in drei Varianten unterteilt.

1. **Positiver Zusammenhang**

   Je höher oder niedriger der Wert der einen Variable ist, desto höher oder niedriger ist der Wert der anderen Variablen.

2. **Negativer Zusammenhang**

   Je höher oder niedriger der Wert der einen Variable ist, desto niedriger oder höher ist der Wert der anderen Variablen.

3. **Kein Zusammenhang**

   Die Höhe die Werte der Variable verändern sich nicht miteinander und haben damit keinen Einfluss auf die Änderung der anderen Variablen.

Die Korrelation wird mit Hilfe der aufgezeigten Formel berechnet und das Ergebnis repräsentiert den Korrelationskoeffizient [Humboldt-Universität, 2019].

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{(\sum_{i=1}^{n}(x_i - \bar{x})^2)}(\sum_{i=1}^{n}(y - \bar{y})^2)}$$

Mit dem Korrelationskoeffizienten kann eine Aussage über den Stärkegrad einer linearen Beziehung zweier Werte getroffen werden, welcher sich in einem Wertebereich zwischen -1 und 1 befinden kann.

Einteilung des Korrelationskoeffizienten ($\boldsymbol{r}$) erfolgt nach [Cohen, 1988, S. 83].

- $\boldsymbol{r} = 0$ --> keine
- $\boldsymbol{r} = 0{,}1$ bis $\boldsymbol{r} = 0{,}3$ --> gering bis moderat
- $\boldsymbol{r} = 0{,}3$ bis $\boldsymbol{r} = 0{,}5$ --> moderat bis groß
- $\boldsymbol{r} > 0{,}5$ --> groß

Wenn der Betrag des Korrelationskoeffizientens unter 0,1 liegt, gibt es keine Beziehung zwischen zwei Werten. Ab einem Betrag von 0,1 spricht man von einer geringen Korrelation und ab einem Betrag von 0,3, spricht man von einer moderaten Korrelation. Liegt der Betrag oberhalb von 0,5, geht man nach Cohen von einem großen Zusammenhang aus. Anhand dieses Bewertungsschemas werden die Messergebnisse in dieser Arbeit bewertet.

Zu erwähnen ist, dass eine Korrelation zwar sehr deutlich zeigen kann, dass zwei Variablen gemeinsam variieren bzw. irgendwie zusammenhängend zu scheinen, impliziert jedoch keine Kausalität. Die Berechnung des Korrelationskoeffizientens erlaubt nicht zu bestimmen, ob zwei Variablen wirklich einen Einfluss aufeinander haben oder ob es eine weiter bisher nicht untersuchte Variable gibt, von der der Einfluss ausgeht [Bittmann, 2016].

Der Korrelationskoeffizient in dieser Arbeit ist als notwendige Bedingung für einen kausalen Zusammenhang zu betrachten und dient der Quantifizierung der Messergebnisse.

3 Konzeption

Der zu entwickelnde Microservice ist ein neues Feature, welches eine eigenständige Fachlichkeit besitzt und zu keinem anderen Service oder Domäne gehört. Damit ist sichergestellt, dass durch den Einbau des Microservice, kein bestehender Code geändert wird. Im Falle des hier zu entwickelnden Services, entspricht in seiner Aufgabe diesen Voraussetzungen, da es ein eigenständiger Service ist, welcher nur indirekt in die Prozesse der anderen Services eingreift. Nicht zuletzt ist diese Architektur gewählt worden, da bei Bringmeister alle Microservices nach dem Prinzip des Domain-driven Designs aufgeteilt sind. Daher war im Grunde das Architektur Muster schon weitestgehend vorgegeben.

Der zu Microservice muss zwei grundsätzliche Aufgaben erfüllen:

1. Die Durchlaufzeiten des Verarbeitungsprozesses überwachen und messen.
2. Die gemessenen Daten zum Herunterladen bereitstellen.

Dieses Kapitel soll einen Überblick über die Idee und die Herangehensweisen aufzeigen, wie dies Umgesetzt werden soll.

Zunächst wird die Servicestruktur kurz erklärt und wo im Verarbeitungsprozess der neue Microservice eingebunden werden soll. Danach folgt eine Darstellung des Konzeptes als Sequenzdiagramm und im Anschluss, wie der Ablauf der Verarbeitung des Microservice angedacht ist.

Dabei wird gezeigt, wie und wo die Messungen in der Servicelandschaft des Fachbereichs Connect-Backend stattfinden und eine kurze Erklärung, wie die Ergebnisse der Messungen berechnet werden. Zu guter Letzt wird noch auf die Auswertung der Ergebnisse eingegangen.

3.1 Vorstellung der Servicearchitektur

Die Service Struktur des Online-Shops der Bringmeister GmbH ist in zwei Softwareentwicklungs-Domänen (Fachbereiche) aufgeteilt, die aus den Fachbereichen Frontend und Backend bestehen, wobei der Fachbereich Backend nochmal in zwei Unterfachbereiche (Connect- und Shop-Backend) aufgeteilt ist. Diese sind für sich eigene Service-Architekturen mit unterschiedlichen Entwicklungssprachen und eigenen Datenverarbeitungssystemen. Im nachfolgenden sind diese drei Fachbereiche aufgelistet und deren Aufgabenbereiche kurz erklärt.

Frontend

- Ist für die Visualisierung der Daten für den Konsumenten (User) verantwortlich. In diesem Bereich werden die Web-Applikationen und Mobile-Applikationen entwickelt und bereitgestellt.

Shop-Backend

- Ist für die Verarbeitung der gesamten sensiblen Kundendaten und Bestellungs-abrechnungen verantwortlich.

Connect-Backend

- Ist für die Verarbeitung der kompletten Produktdaten verantwortlich. Die Kommunikation mit und zwischen den einzelnen Services wird über REST-APIs realisiert.

3.2 Verarbeitungsprozess der Produktdaten

In diesem Kapitel wird der bestehende Verarbeitungsprozess dargestellt und erklärt. Dieser Prozess findet zwischen dem Lager System (Startpunkt) und Algolia[1] (Endpunkt) statt. Im Anschluss wird aufgezeigt und erläutert, wie der Microservice in den Prozess eingebunden werden soll und wo die Messpunkte angelegt werden könnten, um die Messdaten zu generieren.

3.2.1 Darstellung des Verarbeitungsprozesses

In [Abbildung 1] ist der betrachtete Verarbeitungsprozess dieser Arbeit grafisch dargestellt, welcher überwacht werden soll. Die Namen der einzelnen Abschnitte sind abgewandelt und repräsentieren deren Funktionalität. Anschließend wird dieser Prozess genauer erklärt.

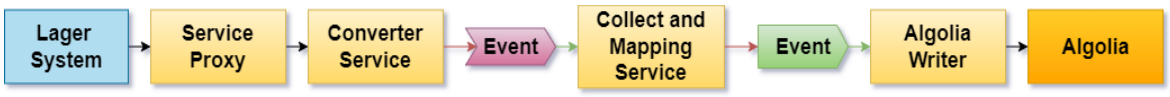


Abbildung 1. Verarbeitungsprozess zwischen Lager und Algolia

[1] Algolia ist eine Software-as-a-Service Technologie, die eine intelligente und schnelle Suche für Unternehmen bereitstellt.

Der Prozess beginnt im Lager System, von dem die Änderungen von Preis-, Produkt- oder Verfügbarkeitsdaten als XML[2]-Datei an den Service Proxy über eine SOAP[3]-API übermittelt werden. Diese XML-Dateien beinhalten eine Ansammlung von mindestens eins bis beliebig viele Produktdaten. Der Service Proxy befindet sich in der Service-Architektur des Fachbereichs Connect-Backend, wo diese Daten über die genannte SOAP-API angenommen und verarbeitet werden. Im Service Proxy werden diese Daten archiviert und an den Converter Service weitergeleitet. Dort werden die Daten aufbereitet und als Event weiter an den Collect and Mapping Service geschickt.
Dieser Service ist der Sammelpunkt, an dem alle relevanten Produktdaten aus verschiedenen Services eintreffen und gesammelt werden. Nach dem der Collect and Mapping Service dieses Event erhalten hat, wartet dieser, bis alle notwendigen Daten aus den anderen Services eingetroffen sind, die benötigt werden, um ein Event mit allen erforderlichen Produktdaten zu generieren. Anschließend wird das Event an den Algolia Writer Service geschickt und von dort aus nach Algolia geschrieben, womit der der Verarbeitungsprozess abgeschlossen ist.

3.2.2 Einbindung des Microservice in den Verarbeitungsprozess

In der nachfolgenden [Abbildung 2. Verarbeitungsprozess zwischen Lager System und Algolia mit eingebundenen Microservice Chronos wird zunächst aufgezeigt, wie der Microservice Namens Chronos[4] in den Verarbeitungsprozess aus [Abbildung 1] eingebunden wird. Im Anschluss wird kurz erläutert, wo die Messpunkte für die Messung des Verarbeitungsprozesses angesetzt werden sollen.

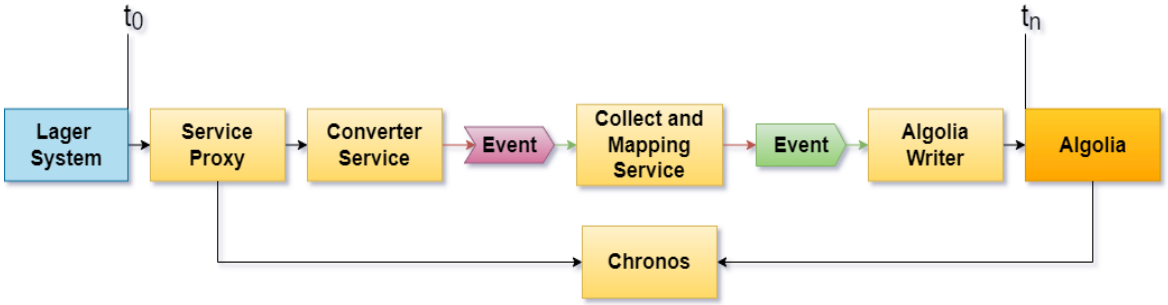


Abbildung 2. Verarbeitungsprozess zwischen Lager System und Algolia mit eingebundenen Microservice Chronos

[2] XML ist eine Definition, wie Daten in einer Textdatei strukturiert gespeichert werden.
[3] SOAP ist ein Internetprotokoll was mit dem W3C-Standards kompatibel ist und bietet die Möglichkeit der Kommunikation zwischen Client und Server. [Adams et al. (2012)]
[4] Chronos ist der Name der Microservice und ist in Anlehnung an den griechischen Gott der Zeit gewählt, da die Funktionalität des Service zur Erfassung von Durchlaufzeiten genutzt wird.

Der Microservice Chronos, der die Grundlage dieser Arbeit ist, soll wie in [Abbildung 2] gezeigt, in Verarbeitungsprozess eingebunden werden. Dabei ist es wichtig, dass der Service separat und nicht in den Verarbeitungsprozess eingebunden wird, da dieser sonst die Messungen verfälschen könnte. Der Service soll die Durchlaufzeit (s. Kapitel [2.5.1]) messen, die während des Verarbeitungsprozesses durch die aufgezeigte Service-Architektur anfällt.

Die Messung wird wie in [Abbildung 2] an den Messpunkten $\mathbf{t_0}$ und $\mathbf{t_n}$ durchgeführt. Der erste Messpunkt, der betrachtet wird, ist das Lager System. Die dort versendeten XML-Dateien werden mit einem Zeitstempel über deren Erstellung versehen, welcher als Startpunkt $\mathbf{t_0}$ der Messung betrachtet wird. Die Produktdaten, die nach Abschluss des Verarbeitungsprozesses nach Algolia geschrieben werden, stellen den Endpunkt $\mathbf{t_n}$ der Messung dar. Mittels dieser zwei Zeiten kann im Anschluss das gesuchte Delta $\mathbf{\Delta t}$ berechnet werden, was die Durchlaufzeit repräsentiert.

Darüber hinaus soll der Service so flexibel gestaltet werden, dass weitere Teilstrecken hinzugefügt werden können, um weitere Messpunkte für eine detailliertere Analyse des Verarbeitungsprozesses zu erhalten.

3.2.3 Konzeptionierung des Microservice als Sequenzdiagramm

Die folgende [Abbildung 3] zeigt das konzeptionelle Sequenzdiagramm und wie der grobe Ablauf des Verarbeitungsprozesses des eingebundenen Microservice angedacht ist.

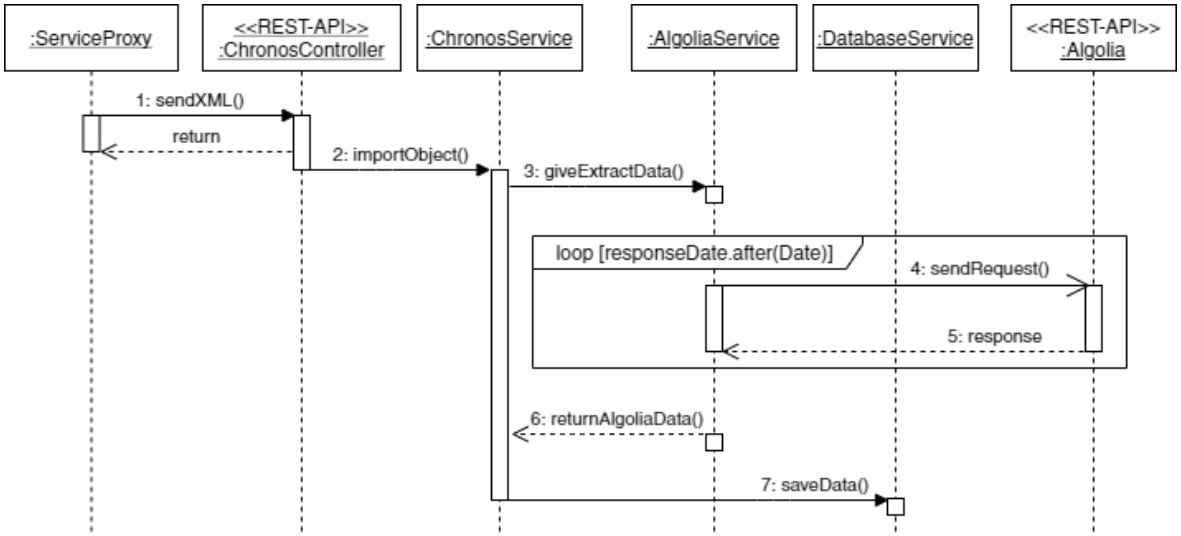


Abbildung 3. Allgemeines Konzept als Sequenzdiagramm

Der Service Proxy ist der Einstiegspunkt in der Service-Architektur, an den eingegriffen werden kann. Dort muss zunächst eine Möglichkeit geschaffen werden, die XML-Dateien mit den Produktdaten aus dem Lager System an den Microservice Chronos zu übermitteln.

Dies kann durch den Einbau einer zusätzlichen Route im Service Proxy realisiert werden, da dieser die XML-Dateien bereits an den Counter Service weiterleitet. Dafür wird im Chronos Service eine REST-API erstellt, mit der die weitergeleiteten XML-Dateien empfangen werden können.

Nach Erhalt der XML-Datei werden diese zuerst in Java-Objekte umgewandelt, bevor in einem weiteren Prozess die benötigten Daten aus diesen Objekten extrahiert und gespeichert werden. Die Daten, die aus der XML-Datei ausgelesen werden, sind der Zeitstempel, sowie die SKUs[5] mit den dazugehörigen Standorten. Der Zeitstempel kennzeichnet die Erstellung der XML-Datei. Die SKUs sind die Primärschlüssel der Produkte und die Standorte repräsentieren das jeweilige Lager, des Aktualisierten Produkts gilt.

Im nächsten Schritt wird mittels der gespeicherten Daten eine wiederholte Abfrage an Algolia gestartet, mit der das aktualisierte Produkt abgefragt wird. Algolia stellt dafür eine API bereit, mit der eine Suche nach gewünschten Produkten erfolgt. Der Zeitstempel des erhaltenen Produkts muss im Anschluss überprüft werden, ob dieser älter ist als der Zeitstempel aus der XML-Datei. Diese Abfrage wird so lange wiederholt, bis das zuvor beschriebene Szenario erfüllt ist oder eine eingestellte Zeitspanne überschritten wird.

Um dies zu realisieren, muss ein intervallgesteuertes Abfrageverfahren entwickelt und implementiert werden. Dieses Verfahren muss in der Lage sein, die folgenden drei Punkte zu erfüllen:

- Asynchrone Abfragen an Algolia, um den Prozess des Services nicht aufzuhalten oder zu unterbrechen.
- Einstellbare Zeitspanne, wie lange der Abfrageprozess an Algolia ausgeführt wird, bevor der Prozess abbricht.
- Einstellbares Abfrageintervall, in welchem zeitlichen Abstand die Abfragen an Algolia stattfinden soll.

Nach Erhalt des aktualisierten Produkts von Algolia, werden alle benötigten Daten in einem neuen Objekt in einer eigenen Datenbanktabelle gespeichert. Anschließend werden aus diesem Objekt die Zeitstempel ausgelesen und die Differenz berechnet, was die Durchlaufzeit des Verarbeitungsprozesses repräsentiert. Bereitstellung der Daten

Die zweite Aufgabe, die der Mircoservice Chronos erfüllen soll, ist die Bereitstellung der gemessen Daten.

[5] Die SKUs (Stock Keeping Unit) sind die Produktnummern, die von der Firma Bringmeister GmbH selbst vergeben werden.

Dies soll über eine REST-API bewerkstelligt werden, mit der sich die gesammelten und gemessen Daten als CSV-Datei herunterladen lassen. Diese CSV-Datei kann daraufhin mit einem externen Programm (z.B. Excel) geöffnet werden. Mittels der dort vorhandenen Funktionen können die Daten analysiert und ausgewertet werden. Zusätzlich können diese Daten als Diagramme grafisch aufbereitet und visualisiert werden.

3.3 Messungen

Im Nachfolgenden wird das Konzept erläutert, wie die Messungen durchgeführt werden.

Das Ziel ist, die Durchlaufzeit des Verarbeitungsprozesses anhand der Produktdaten zu ermitteln, die durch die Service-Architektur verarbeitet werden. Um Messungen durchführen zu können, werden Zeitdaten benötigt. Diese werden auf Grundlage von Zeitstempel erhoben, welche aus den Daten der einzelnen Systeme ermittelt werden. Der Zeitstempel aus der XML-Datei des Lager Systems ist die erste Zeitangabe, die zur Verfügung steht und damit den Start der Messung darstellt. Der Zeitstempel aus dem abgerufenen Algolia Produkts, ist der letzte Punkt des Verarbeitungsprozesses und repräsentiert damit das Ende der Messung.

Nach der Ermittlung dieser beiden Zeiten soll die Differenz zwischen den beiden Zeitstempeln berechnet werden. Diese ermittelte Zeit ist dann die gemessene Durchlaufzeit, die eine entscheidende Messgröße dieser Arbeit darstellt.

4 Implementierung

Der zu entwickelnde Microservice Chronos muss zwei grundsätzliche Aufgaben erfüllen:

- Die Durchlaufzeiten des Verarbeitungsprozesses überwachen und messen.
- Die gemessenen Daten aufbereiten und zum Herunterladen bereitstellen.

Im Folgenden werden die verschiedenen Prozesse mit deren Umsetzung erläutert und die daraus resultierenden Lösungen begründet. Dabei ist zu beachten, das externe System involviert sind, die nicht mit dem hier zu entwickelten Chronos Service überwacht und verfolgt werden können.

4.1 Annahme der Produktdaten und Konvertierung in Java Objekte

Die Kommunikation zwischen der Außenwelt, in der sich das Lager System befindet, und der Innenwelt, die den Service Proxy der Connect-Backend Service-Architektur beinhaltet, läuft über eine SOAP-API per HTTP-Request. Über diesen Request werden die Produktdaten als XML-Datei verschickt. Der Service Proxy ist der Einstiegspunkt, um die vom Lager System gesendeten XML-Dateien zu erhalten. Dafür ist im Service Proxy eine Route eingebaut, über die die Daten an den Chronos Service geschickt werden.

Die im Chronos Service implementierte REST-API stellt den Zugriffspunkt für die ankommenden Daten aus dem Proxy Service bereit. Die XML-Dateien werden nach dem Empfang in Java-Objekte konvertiert (s. Kapitel [4.1.1]). Aus den generierten Java-Objekten werden in einer separaten Mapper-Klasse die relevanten Daten extrahiert. In dieser Mapper-Klasse wird die Struktur des zuvor generierten Objektes durchsucht und die ausgelesenen Daten in ein neues Datenobjekt geschrieben. Alle in der XML-Datei enthaltenen Produktdaten werden als Liste zurückgegeben und für eine bestimmte Zeit in einem Zwischenspeicher (s. Kapitel 4.1.2]) vorgehalten, bis entweder die Daten von dort ausgelesen oder eine eingestellte Zeit abgelaufen ist. Beides führt zum Löschen der Daten.

4.1.1 Konvertierung der XML-Dateien in Java Objekte

Um den XML-Input in Java Objekte zu konvertieren, ist die Jakarta XML Binding (kurz JAXB) Bibliothek [Oracle, 2016] verwendetet worden. Damit lässt sich aus einer XSD[6]-Datei die dazugehörige Java-Klasse automatisch erstellen, um danach mittels einer konkreten XML-

[6] XSD ist die Definition oder auch Schemasprache, die den Inhalt und Aufbau einer konkreten XML-Datei beschreibt.

Datei die Java-Objekte aus den zuvor erwähnten Klassen zu generieren [Ullenboom, 2021, Kapitel 16.4] . Die Vorteile sind zum einen, nicht selbst für das XML-Handling verantwortlich zu sein und zum anderen stellt diese Bibliothek eine zuverlässige und erprobte Lösung dar, wodurch Fehler vermieden werden.

4.1.2 Zwischenspeichern der extrahierten Daten

Zum Speichern der eingehenden Daten bieten sich zwei Lösungswege an:

1. Das Persistieren der Daten in einer Datenbank (z.B. DynamoDB oder MySQL).
2. Das Vorhalten der Daten in einem (In-Memory) Cache (z.B. Spring Cache oder Guava Cache).

Für die Implementierung des Chronos Services ist das Vorhalten der Daten in einem Cache als Lösungsweg aus den folgenden Gründen gewählt worden.

Der Cache hat den Vorteil, dass keine Datenbank angebunden werden muss, da dies zu mehr Kosten und mehr Code geführt hätte. Ein weiterer Grund für den Cache liegt darin, dass die Daten nicht dauerhaft gespeichert werden, was in diesem Fall auch nicht notwendig ist. Die Daten müssen nur für einen gewissen Zeitraum vorgehalten werden, bis sie abgerufen und weiterverarbeitet werden.

Die vorher beschriebenen Punkte ließen sich mittels der Guava: Google Core Libraries for Java Bibliothek [Kluever, 2021] realisieren. Damit lassen sich Daten, wie im Folgenden [Listing 1] gezeigt, zwischenspeichern.

Die Implementierung des Guava Caches ist in einer separaten Klasse gekapselt.

Listing 1. Buffer-Klasse zum Zwischenspeichern der einkommenden Produktdaten

```
1 class StockXmlBuffer(
2     private val chronosProperties: ChronosProperties
3 ) {
4
5     private val cache = CacheBuilder
6         .newBuilder()
7         .expireAfterWrite(chronosProperties.cacheTime, SECONDS)
8         .build<String, StockImportData>()
9
10    fun buffer(objectId: String, data: StockImportData) {
11        cache.put(objectId, data)
12    }
13
14    fun getFromBuffer(objectId: String): StockImportData? {
15        return cache.getIfPresent(objectId)
16    }
17 }
```

- In Zeile 5 bis 8 wird eine Instanz des Caches über das Builder-Pattern [Gamma, 2015, S. 101] erzeugt. Dabei wird in Zeile 7 die Lebensdauer eines Eintrags im Cache festgelegt. Nach Ablauf dieser Zeitspanne werden Einträge im Cache automatisch gelöscht.
- Zeile 10 bis 12 beschreiben eine Funktion zum Befüllen des Caches.
- Zeile 14 bis 16 beschreiben eine Funktion zum Auslesen des Caches.
- Konfigurationsparameter werden als eigenes Objekt in Zeile 2 an die Klasse übergeben. Diese beinhalten etwa die Lebensdauer für Cache-Einträge.

4.2 Konfiguration und Umsetzung des Abfrageprozesses an Algolia

In diesem Abschnitt wird ein Kernpunkt dieser Arbeit behandelt. Es wird dafür zunächst darauf eingegangen, welche Verfahren es gab, um Produktdaten von Algolia zu erhalten. Anschließend wird beschrieben, wie die Kommunikation mit Algolia umgesetzt worden ist und welche Konfigurationsschritte dafür notwendig waren. Danach wird die Umsetzung des ausgewählten Verfahrens aufgezeigt und beschrieben.

4.2.1 Auswahl und Umsetzung des Kommunikationsverfahrens

Im Nachfolgenden werden die drei üblichen Kommunikationsverfahren (s. Kapitel [2.2]) in Hinblick auf den in dieser Arbeit behandelten Anwendungsfall bewertet. Anschließend wird das Verfahren beschrieben, welches auf Grundlage der Bewertung ausgewählt Umsetzung worden ist.

Bei einem Push-Verfahren würde Algolia die Produktdaten nach deren Aktualisierung automatisch an den Service schicken. Dies wäre die ideale Lösung, da es die genaueste Zeit über den Abschluss des Verarbeitungsprozesses liefern würde. Da dies von Algolia aber leider nicht unterstützt wird, konnte dieses Verfahren nicht genutzt werden.

Bei einem Pull-Verfahren würde Algolia den Service darüber informieren, das sich Produktdaten geändert haben. Diese Funktion wird von Algolia ebenfalls nicht unterstützt, da es nur eine API bereitstellt, über die nach Produkten bei Algolia gesucht und diese abgerufen werden können.

Aus diesem Grund fiel die Entscheidung für die Umsetzung auf das Polling-Verfahren, dabei werden Produkte wiederholt abgefragt und überprüft, ob sich etwas in den Produktdaten geändert hat. Dies ist mittels der Java Bibliothek Awaitility [Haleby, 2010] umgesetzt worden. Diese bietet eine breite Auswahl an Einstellungsfunktionen, welche für die Umsetzung vorteilhaft sind.

4.2.2 Konfiguration zur Kommunikation mit der Algolia-API

Im Folgenden werden die Schritte für die Konfiguration zum Abfragen von Produktdaten bei Algolia beschrieben. Die Abfrage ist so konfiguriert, dass über einen Index nach Produkten gesucht werden kann. Der Indize war im Vorfeld schon konfiguriert und repräsentiert die ObjectID. Diese setzt sich aus der Produktnummer (SKU) und der Standortnummer (StoreID) zusammen, welche durch einen Unterstrich getrennt sind:

Beispiel:

```
123456_1 -----> ObjectID
|……..|_|.|
 SKU      StoreID
```

Die ObjectID ist der Schlüsselwert, über den in Algolia nach dem gewünschten Produkt gesucht wird. Dafür ist zunächst eine Konfigurations-Klasse geschrieben worden, in dem der Search Client von Algolia implementiert ist. Darin werden die hinterlegten Zugangsdaten geladen, die aus einer Anwendungs-ID und einem API-Schlüssel bestehen und für die Anmeldung bei Algolia verwendet werden. Die Anwendungs-ID ist die Identifikationsnummer, welche bei der Erstellung eines Algolia Kontos sowohl erzeugt als auch vergeben wird.

Im nächsten Schritt erfolgte die Implementierung einer Algolia Gateway[7]-Klasse, die eine Verbindung mit dem Algolia-Client aufbaut, mit dem die Abfragen an Algolia ausgeführt werden. In dieser Klasse wird der Index und der zuvor erstellte Algolia Search Client importiert. Der Index ist die Instanz von Algolia, in der gesucht wird. Es gibt zwei Instanzen, die zur Verfügung standen, die für die Testdaten und die für die Produktivdaten. Die Anfrage an Algolia wird über eine Funktion durchgeführt, die mittels der übergebenen ObjectID das Produkt zurückgibt, wenn es gefunden wird. Wenn es nicht gefunden wird, tritt der Fehler Code 404 auf, was eine Ausnahme (engl. Exception) auslöst. Damit ist die Konfiguration für die Kommunikation mit Algolia bereitgestellt.

[7] Ein Gateway ist ein Vermittler, der die Kommunikation zwischen zwei Systemen herstellt, die nicht kompatibel zueinander sind.

4.2.3 Intervallgesteuerte Abfrage mittels Polling-Verfahrens

Die Abbildung [**Fehler! Verweisquelle konnte nicht gefunden werden.**] zeigt die I mplementierung der Einstellmöglichkeiten des Polling-Verfahrens. Der Ablauf der Funktion des Verfahrens wird im Nachhinein erläutert.

Listing 2. Funktion des Polling-Verfahrens an Algolia

```
 1 fun pollAlgolia(product: StockImportData): ChronosData {
 2     val objectId = product.sku + "_" + product.storeId
 3     var numberOfPolls = 0
 4     return Awaitility
 5         .await()
 6         .pollInSameThread()
 7         .atMost(chronosProperties.atMost)
 8         .with().pollDelay(chronosProperties.delay)
 9         .and().pollInterval(chronosProperties.interval)
10         .untilNotNull {
11             //...
12 }
```

- Zeile 1 zeigt die Funktionsdeklaration, sowie deren Namen, ihren Übergabeparameter und den Rückgabetype.
- Zeile 2 zeigt die Initialisierung einer unveränderlichen Variable für die ObjectID.
- Zeile 3 zeigt die Initialisierung einer veränderlichen Variable mit dem Wert null.
- Zeile 4 bis 10 zeigen die Eigenschaften der Funktion, die in den Zeilen 7 bis 9 übergeben wird. Diese werden in einer separaten Klasse gesetzt.
  - Zeile 4 zeigt die Rückgabe der Funktion, die mittels der Awaitility Funktion erzeugt wird.
  - Zeile 5 definiert die Erstellung einer Await-Anweisung.
  - Zeile 6 definiert einen Thread[8] in den die Funktion ausgelagert wird.
  - Zeile 7 definiert den Zeitraum, wie lange die Funktion ausgeführt wird, bis sie beendet wird.
  - Zeile 8 definiert eine Verzögerung, nach der die erste Abfrage ausgeführt wird.
  - Zeile 9 definiert die Wiederholungsrate der Funktion, in welchen Abständen diese ausgeführt werden.
  - Zeile 10 definiert, dass die Funktion solange ausgeführt wird, bis der Rückgabewert nicht null ist.

[8] Ein Thread ist ein Teil eines Prozesses, der in einem parallelen Ausführungsstrang auslagert und verarbeitet wird.

4.2.4 Ablauf des Abfrageprozesses der Produkte von Algolia

In diesem Abschnitt wird, wie in Abbildung 4 zu sehen ist, der Ablauf des Abfrageprozesses der Produkte von Algolia aufgezeigt und im Anschluss beschrieben.

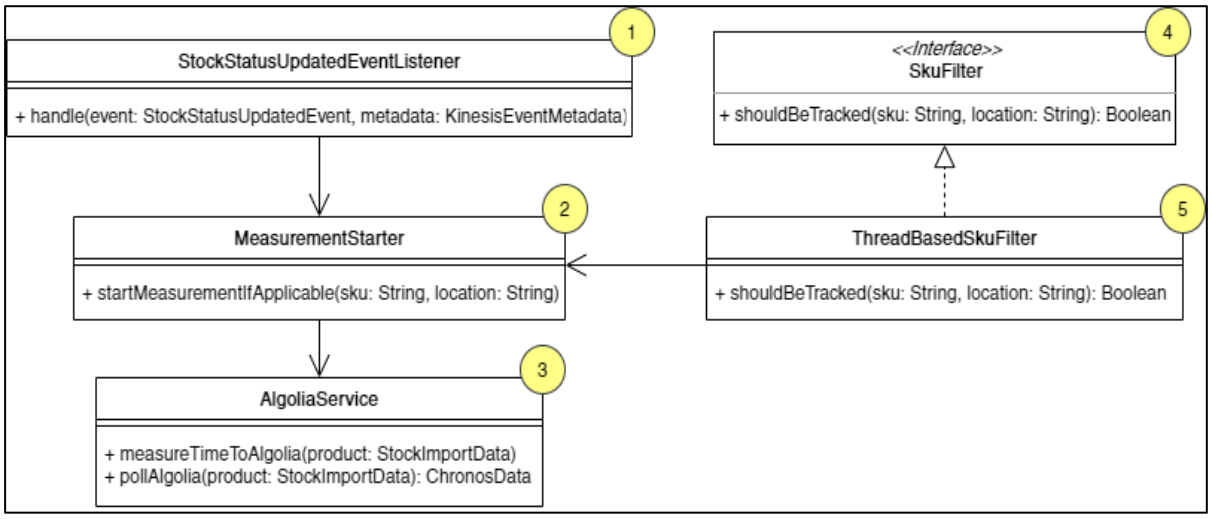


Abbildung 4. Klassendiagramm des Abfrageprozesses

Dafür ist zunächst ein Event-Listener (s. Abbildung 4, Punkt 1]) erstellt worden, der die Stock-Events (Verfügbarkeits-Events) bekommt, wenn diese vom Converter Service (s. [Abbildung 2]) verschickt werden.

Dieser Listener übermittelt daraufhin die notwendigen Daten (SKU und Standort), die aus dem Stock-Event extrahiert werden, an die Funktion *startMeasurementIfApplicable* (s. Abbildung 4, Punkt 2]), die den Prozess zur Abfrage der Produkte von Algolia anstößt. Diese Funktion wird nur ausgeführt, wenn das Ergebnis der internen Bedingungen „wahr“ ist, was anhand zweier boolescher Werte überprüft wird. Dabei steht der erste Boolean-Wert für einen festen Wert, der für den Zeitraum der Messungen auf „wahr“ gesetzt wird. Damit besteht die Möglichkeit den Service an- und auszuschalten.

Mit dem zweiten Boolean-Wert werden die erhaltenen Listener-Daten mittels der Interface-Funktion (s. Abbildung 4, Punkt 4]) in einer eigenen Thread-Klasse (s. Abbildung 4, Punkt 5]) ausgewertet. Das Interface ist am Anfang der Implementierung des Services eingebaut worden, damit erst einmal nur ausgewählte Produkte überwacht werden.

Die Thread-Klasse (s. Abbildung 4, Punkt 5]) hat diese Funktionalität später ersetzt. Dabei werden die aktiv laufenden Threads ausgelesen und mit einem fest eingestellten Wert verglichen, welcher die Anzahl der zur Verfügung stehenden Threads repräsentiert. Danach wird überprüft, ob der Wert größer als die Anzahl der aktiven Threads ist.

Anschließend wird aus dem übergebenen Parameter eine Objekt ID generiert, mit der die zwischengespeicherten Daten (s. Kapitel [4.1.2]), aus dem Cache abrufen werden. Die

Daten aus dem Cache werden dann noch einmal überprüft, ob diese Daten vorhanden sind, da sonst eine Log-Warnung ausgegeben wird. Die Verarbeitung der Cachedaten ist in eine separate Klasse (s. Abbildung 4, Punkt 3]) ausgelagert worden, in der die asynchrone Verarbeitung (s. Kapitel [2.4.1]) mittels der Spring Bibliothek [Spring, 2002] implementiert ist. Dies hat den Vorteil, dass die ausführende Funktion nur mit der @Async Annotation gekennzeichnet wird, wodurch Spring die parallele Verarbeitung in separaten Threads abarbeitet. Der auszuführende Code wird dafür in eine try-catch-Anweisung gekapselt, um bei einer Fehlfunktion den Abbruch der Programmausführung zu verhindern. Im try-Block ist die Verarbeitung der Daten eingebaut, mit denen die intervallgesteuerten Funktion *pollAlgolia* aufruft wird, die mittels Polling-Verfahrens (s. Kapitel [4.2.3]) das aktualisierte Produkt von Algolia abfragt.

Danach werden die Produktdaten und Messdaten jeweils in einer eigenen Datenbanktabelle gespeichert. Dies hat den Vorteil, dass es sich bei diesen Daten um die Ausgangsdaten handelt, die später noch für die Berechnung der Messung verwendet werden. Die Produktdaten beinhalten die extrahierten Daten aus der XML-Datei und den Zeitstempel des abgefragten Produkts von Algolia. Die für die Messungen gespeicherten Daten werden im Kapitel [4.3.2] zur Berechnung der Messung beschrieben. Der catch-Block besteht aus einer Ausnahme, die im Fehlerfall ausgelöst wird, wodurch eine Log-Warnung ausgegeben wird, in der Informationen über den Fehler enthält sind.

4.3 Berechnung und Export der Messdaten

Der folgende Abschnitt zeigt die Berechnung der Messdaten für die Durchlaufzeit. Es wird außerdem auf die Bereitstellung der Daten als CSV-Dateien eingegangen.

Weiterhin wird erläutert, wie der Vorgang zur Berechnung der Messdaten für die Durchlaufzeit des Verarbeitungsprozesses umgesetzt ist. Dabei wird darauf eingegangen, wie die berechneten Daten als CSV-Datei konvertiert werden und welche Funktionalität zum Herunterladen dieser CSV-Dateien umgesetzt worden ist.

4.3.1 Erhebung der Messdaten

Die ermittelten Durchlaufzeiten sind anhand der gesetzten Zeitstempel, wie in (s. Kapitel [4.3.2]), die aus den erhaltenen und abgefragten Daten der einzelnen Systeme berechnet worden sind. Der Chronos Service speichert die folgenden drei Zeitstempel als Messpunkte während eines Verarbeitungsprozesses, wie in [Abbildung 5]:

- t_0 = Erstellungszeit einer XML-Datei aus dem Lager System.
- t_1 = Ankunftszeit der Daten aus dem Lager System im Chronos Service.
- t_2 = Ankunftszeit eines Produkts von Algolia im Chronos Service.

Die Differenz der zuvor aufgeführten Zeiten zwischen $\mathbf{t_0}$ und $\mathbf{t_2}$ repräsentieren die ermittelte Durchlaufzeit durch die Service-Architektur.

Abweichend als ursprünglich konzipiert (s. Kapitel [3.2.2]), werden drei statt zwei Zeitstempel für die Durchlaufzeiten zwischen den aufgeführten Prozessabschnitten berechnet. Die dritte Zeit $\mathbf{t_1}$ ist eingeführt worden, um die starken Schwankungen der Durchlaufzeiten zu analysieren, die bei den ersten Betrachtungen der Messergebnisse auffielen.

1. Lager System --> Algolia => $t_2 - t_0$ => Prozessabschnitt 1
2. Lager System --> Chronos Service => $t_1 - t_0$ => Prozessabschnitt 2
3. Chronos Service --> Algolia => $t_2 - t_1$ => Prozessabschnitt 3

Die Messergebnisse, die im folgenden Kapitel [5.1.3] visuell aufgeführt werden, repräsentieren die durchschnittlichen Durchlaufzeiten der Verarbeitungsprozesse der zuvor aufgeführten Prozessabschnitte. Diese Durchschnittswerte berechnen sich anhand der Fünf-Minuten-Intervalle und des arithmetischen Mittelwerts. Damit sollten die Schwankungen der absoluten Messwerte rausgemittelt werden, um eine repräsentativere Aussage über die Durchlaufzeiten treffen zu können.

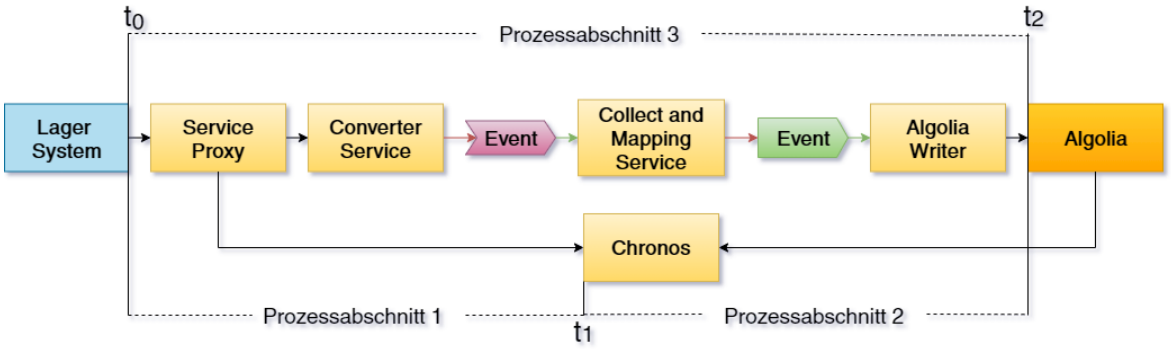


Abbildung 5. Verarbeitungsprozess zwischen Lager System und Algolia mit dem eingebundenen Microservice Chronos und erstellten Messpunkte

4.3.2 Berechnung der Messdaten

Die Ermittlung der Durchlaufzeit beginnt mit der Zeitmessung, welche mit der Erstellungszeit einer XML-Datei aus dem Lager System angestoßen wird. Dafür werden die relevanten Daten aus dieser Datei extrahiert und diese zusammen mit dem Zeitstempel der aktuellen Zeit gespeichert. Dieser zusätzliche Zeitstempel repräsentiert die Ankunftszeit der Daten aus dem Lager System im Chronos Service. Daraus lässt sich die Durchlaufzeit in zwei Messstrecken aufteilen, die getrennt voneinander betrachtet und etwaige Abweichungen besser analysiert werden können. Die Messung des Prozessabschnitts zwischen dem Lager System und dem Chronos Service ist damit abgeschlossen. Mit den extrahierten Daten wird mittels des Abfrageprozesses die aktualisierten Produkte von Algolia abgefragt und nach dessen Erhalt mit einem Zeitstempel der aktuellen Zeit gespeichert. Damit endet der Messzyklus der Durchlaufzeiten für Produktdaten in den jeweiligen Prozessabschnitten.

Da es sich bei den Durchlaufzeiten um einzelne Zeiten handelt, die je nach auftretender Last (s. Kapitel [2.3]) stark variieren können, werden diese in Intervallen gespeichert. Die Größe dieser Intervalle ist auf fünf Minuten eingestellt. Der Grund hierfür ist, dass bei einem Fehlerfall nicht allzu viele Messdaten verloren gehen und um dennoch ausreichend viele Durchlaufzeiten aufzuzeichnen. Diese werden in einer eigenen Datenbanktabelle gespeichert und das Intervall ist als Primärschlüssel festgelegt. Mittels des arithmetischen Mittelwerts werden die durchschnittlichen Durchlaufzeiten eines Intervalls berechnet. Damit wird versucht die zuvor erwähnten Schwankungen der Durchlaufzeiten heraus zu mitteln.

Mit der Implementierung eines weiteren Event-Listeners, der auf alle Events hört, die nach Algolia gesendet werden, ist es möglich eine Aussage über die auftretende Last zu treffen. Die Events, die im Intervall von fünf Minuten eintreffen, werden in einer Counter-Klasse gezählt und in einer eigenen Datenbanktabelle gespeichert. Dabei ist das Intervall wieder als Primärschlüssel festgelegt. Durch die Auswertung dieser Daten, wird überprüft, ob eine Korrelation (s. Kapitel [0]) zwischen den Durchlaufzeiten und Counter-Daten besteht.

Das zuvor erwähnte Intervall ist mittels der in [Listing 3] aufgezeigten Funktion umgesetzt und wird im Anschluss erläutert.

Listing 3. Funktion zum Erstellen des 5 Minuten Intervalls

```
1 fun roundUpToFiveMinute(time: OffsetDateTime): OffsetDateTime {
2         return time
3             .withSecond(0)
4             .withNano(0)
5             .plusMinutes((65L - (time.minute + 1)) % 5 + 1)
6     }
```

Die Funktion zur Erstellung des Intervalls ist in den Zeilen 1 bis 6 zu sehen. Dabei zeigt Zeile 1 den Namen dieser Funktion mit seinem Übergabe- und Rückgabewert, welche beide ein OffsetDateTime Objekt sind, was ein Datum-Zeit-Format ist (s. Kapitel [2.5.3]). Zeile 2 zeigt die Rückgabe der erhaltenen Zeit, die in den Zeilen 3 bis 5 erzeugt wird. In den Zeilen 3 und 4 wird der Wert für Sekunden und Nanosekunden jeweils auf den Wert null gesetzt. Zeile 6 zeigt die Berechnung der Intervallzeit, deren errechneter Wert danach auf die Minuten der übergebenen Zeit addiert wird. Dadurch wird erreicht, dass das Intervall immer den Wert der nächsten fünf Minuten besitzt.

4.3.3 Export der Messdaten

Um die Ergebnisse der gesammelten Daten auswerten und visualisieren zu können, ist ein zusätzlicher REST-Controller erstellt worden, der die Funktionalität zum Herunterladen dieser Daten als CSV-Datei bereitstellt.

Das Zeitintervall (s. Kapitel [2.5.2]) ist auf einen ganzen Tag (24 Stunden) definiert, in dem die Abstände der Messdaten zum Herunterladen aufgeteilt sind. Die Konvertierung der Messdaten in ein CSV-Format konnte mit der Apache Commons CSV Bibliothek [Apache, 2021] realisiert werden. Diese Bibliothek bietet eine Vielzahl an Funktionen, mit der sich eine Schablone für eine CSV-Datei einfach erstellen lässt. Dabei kann zum einen das Format „EXCEL“ eigestellt werden, um die Daten zu konvertieren. Durch die Einstellung eines Semikolons als Trennzeichen kann das Programm Excel anhand der Namensbezeichnung automatisch die Aufteilung der Spalten, richtig umsetzen. Die Header-Funktion dient der Beschriftung der Spalten. Die Bibliothek bietet zudem die Funktion eines CSV-Printers, der mit den getätigten Einstellungen und den übergebenen Daten eine CSV-Datei generiert.

Dies wird im [Listing 4] dargestellt und dessen Funktion im Nachhinein erläutert.

Listing 4. REST-Controller zum Runterladen von CSV Dateien

```
1  class CsvFileController(
2      private val chronosCsvService: ChronosCsvService
3  ) {
4
5      @GetMapping("/download/measurements/{date}")
6      fun getCsvFileOfMeasurements(
7          @DateTimeFormat(iso = DATE) @PathVariable date: LocalDate
8      ): ResponseEntity<String> {
9          val csvAsString = chronosCsvService
10             .getCsvFileOfMeasurementsByInterval(date)
11         return ResponseEntity
12             .ok()
13             .header(
14                 CONTENT_DISPOSITION,
15                 "attachment; filename=measurementsDataBy$date.csv"
16             )
17             .contentType(APPLICATION_OCTET_STREAM)
18             .body(csvAsString)
19     }
20 }
```

Dieser REST-Controller liefert über einen HTTP-GET-Request die Messdaten eines ganzen Tages als CSV-Datei. Ein HTTP- Request an den Chronos Service besteht aus zwei Teilen, wie in Zeile 5 zu sehen ist.

Im ersten Teil befinden sich die Informationen zum Server, wie dem Hostname, dem Pfad und dem Parameterwert. Der Hostname besteht aus der Subdomaine und dem Domainnamen. Der Pfad besteht aus der Route, unter der die aufzurufende Funktion für den darunterliegenden Verarbeitungsprozess zum Erstellen einer CSV-Datei zu finden ist. Als letztes folgt eine Pfadvariable, diese ist der Übergabewert, der für die Verarbeitung des Prozesses erforderlich ist.

Der zweite Teil besteht aus der Übersetzung dieser Pfadvariable, welche als Zeichenkette (String) übergeben wird. Da die Funktion aber ein Datumsformat erfordert, muss dieser String erst formatiert werden. Das passiert in Zeile 7, die mittels einer Formatierung nach dem ISO Stilmuster „Date“, den String in ein LocalDate-Format formatiert. In den Zeilen 9 und 10 wird eine Funktion zur Berechnung der Durchlaufzeiten (s. Kapitel [4.3.2]) aufgerufen. Dieser Funktion wird das zuvor formatierte Datum übergeben.

Nach Abschluss der Verarbeitung wird das Ergebnis als Antwort in Form einer CSV-Datei zurückgegeben und steht zum heruntergeladen bereit. In den Zeilen 11 bis 18 ist der Aufbau der Antwort zu sehen, welche aus den folgenden drei Teilen besteht:

1. Statuscode
2. Header
3. Body

Der Statuscode wird durch die Funktion .ok() bei der erfolgreichen Verarbeitung der Anfrage gesetzt, was den Wert 200 repräsentiert. In Zeile 13 bis 16 wird der Header definiert. Dieser besteht zum einen aus der CONTENT_DISPOSITION Funktion, wie in Zeile 14 zu sehen ist. Diese gibt an, ob der erwartete Inhalt inline im Browser angezeigt wird, als Webseite oder als Anhang heruntergeladen und lokal gespeichert wird. Da es sich um einen Anhang handelt, der zurückgegeben wird, werden diese Informationen mittels des „attachment“ mit dem Namen plus der Dateiendung gesetzt, wie die Zeile 15 zeigt. In Zeile 17 wird der Medientyp des Rückgabewerts gesetzt. In Zeile 18 wird der Body gesetzt, der die zurückgebenden Daten enthält.

4.4 Testimplementierung

Dieser Abschnitt beschreibt die Validierung der Services. In der Softwareentwicklung sind Tests ein wichtiger Bestandteil für ein möglichst reibungslosen Ablauf, da mit ihnen im Vorfeld Fehlerfälle definiert und getestet werden.

Mittels des Mockito Frameworks [Mockito, 2021] werden sogenannte Unit-Tests erstellt. Damit lassen sich die Funktionen der einzelnen Klassen testen, mit denen eine korrekte Ausführung dieser Funktionen sichergestellt wird.

Die Teststruktur ist so aufgebaut, dass für jede Anwendungsklasse eine entsprechende Testklasse erstellt wird. Dabei wird das Verhalten der Funktionen einer Anwendungsklasse überprüft, indem verschiedene Szenarien erstellt und mittels unterschiedlicher Zustände oder Ergebnisse, eine Funktion testen. So wird sichergestellt, ob die Funktion das liefert, was von ihr erwartet wird. Hierfür werden die verwendeten Klassen, Interface und Objekte in den Testklassen gemockt. Damit ist gemeint, dass bestimmte Elemente vorgetäuscht werden, die für den Test zwar benötigt, aber nicht selbst getestet werden.

5 Auswertung

Der in dieser Arbeit entwickelte Microservice stellt die Grundlage zur Überwachung des Verarbeitungsprozess von Produktdaten bereit, um die folgenden zwei Fragen zu beantworten:

1. Wie lange dauert der durchschnittliche Verarbeitungsprozess von Produktdaten?
2. Besteht ein Zusammenhang zwischen der Durchlaufzeit und der Last des Systems?

In den folgendend Kapitel erfolgt die Auswertung, um diese Fragen zu beantworten.

5.1 Darstellung der Ergebnisse

In diesem Abschnitt werden die Ergebnisse aus den heruntergeladenen CSV-Dateien dargestellt, welche mit dem Programm Excel visualisiert und ausgewertet worden sind.

5.1.1 Beschreibung der Diagrammdarstellung

Die Daten aus den CSV-Dateien werden in Diagrammen dargestellt, in denen die unterschiedlichen Prozessabschnitte wie in [Abbildung 5] widergespiegelt werden.

Dabei wird die Durchlaufzeit über einen Zeitraum von 24 Stunden, in Fünf-Minuten-Intervallen dargestellt. Diese sollen dabei helfen, einen allgemeinen Überblick über die durchschnittlichen Durchlaufzeiten zu erhalten, um im Anschluss Rückschlüsse daraus ableiten zu können.

Auf der linken Seite der Y-Achse des Diagramms sind die durchschnittlichen Durchlaufzeiten in Millisekunden aufgeführt. Der Durchschnitt je Fünf-Minuten-Intervall ist als arithmetischer Mittelwert berechnet worden.

Auf der X-Achse ist das Intervall für einen Zeitraum von 24 Stunden eines Tages aufgelistet.

5.1.2 Durchschnittliche Durchlaufzeiten der einzelnen Prozessabschnitte

In diesem Abschnitt wird das Diagramm für einen Tag, Donnerstag, den 03.03.2022 für die durchschnittlichen Durchlaufzeiten der folgenden Prozessabschnitte dargestellt.

- Lager System --> Chronos Service => $t_1 - t_0$ => Prozessabschnitt 1
- Chronos Service --> Algolia => $t_2 - t_1$ => Prozessabschnitt 2
- Lager System --> Algolia => $t_2 - t_0$ => Prozessabschnitt 3

Im Anschluss werden die daraus ermittelten Mittelwerte der Durchlaufzeiten, die Standardabweichungen sowie die minimalen und maximalen Durchlaufzeiten in einer Tabelle zusammengefasst dargestellt.

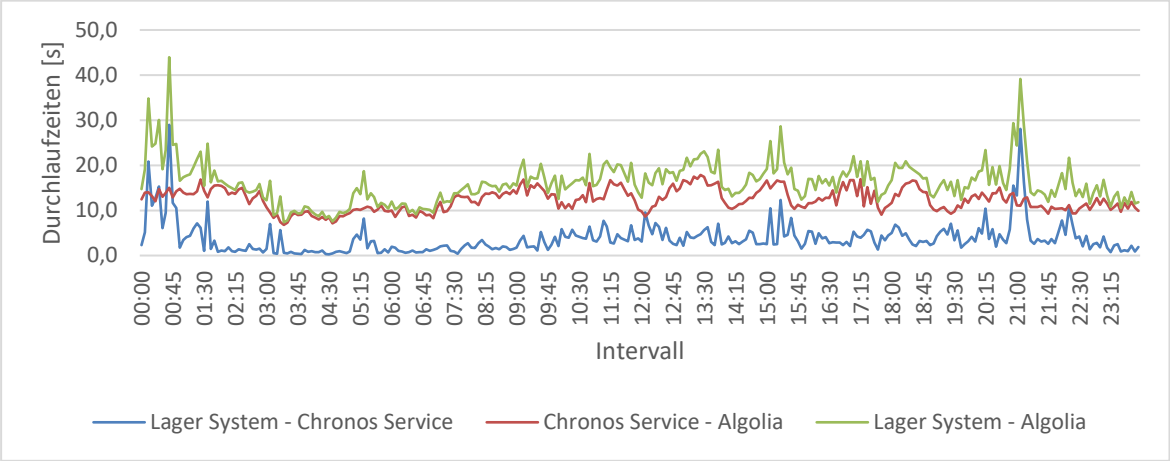


Diagramm 1. Durchschnittliche Durchlaufzeiten der einzelnen Prozessabschnitte

Es ist zu sehen, dass die Durchlaufzeiten zwischen Lager System und Chronos Service starken Schwankungen unterliegen. Bei den Zeiten 0 Uhr bis ca. 0:45 Uhr und bei ca. 21 Uhr sind starke Peaks zu sehen, bei denen die Durchlaufzeiten oberhalb von 35 Sekunden liegen. Die Kurve der Durchlaufzeiten zwischen dem Chronos Service und Algolia zeigt hingegen nicht so starke Schwankungen, aber eine durchschnittliche höhere Durchlaufzeit zwischen 10 Sekunden und 20 Sekunden.

Gut erkennbar ist, dass sich die Kurve der Durchlaufzeiten zwischen dem Lager System und Algolia aus den beiden zuvor diskutieren Kurven zusammensetzt. Diese Kurve unterliegt denselben Schwankungen, wie bei den Durchlaufzeiten zwischen Lager System und Chronos Service.

[Tabelle 1] zeigt die ermittelten Werte sowie das Maximum und das Minimum der Durchlaufzeiten für die drei zuvor erwähnten und dargestellten Prozessabschnitte.

	Prozess-abschnitt 1	**Prozess-abschnitt 2**	**∑ der Prozessabschnitte 1 + 2**	**Prozess-abschnitt 3**
Mittelwert [s]	3,8	12,3	16,1	16,1
Standardabweichung [s]	3,6	2,4	6,0	4,7
Minimaler Wert [s]	0,3	6,8	7,1	7,4
Maximaler Wert [s]	29,0	17,9	46,8	44,0

Tabelle 1. Darstellung der ermittelten Werte der Durchlaufzeiten

Aus der Tabelle wird ersichtlich, dass die Durchlaufzeit zwischen dem Lager System und Algolia im zeitlichen Mittel bei 16,1 Sekunden liegt.

Die Standardabweichung der Durchlaufzeiten zwischen dem Lager System und dem Chronos Service ist fast genauso groß wie der Mittelwert dieses Prozessabschnitts. Dies zeigt sich auch in den der starken Schwankungen dieses Prozessabschnittes in [Diagramm 1].

Anhand der minimalen und maximalen Werte der Durchlaufzeiten wird deutlich, dass die größten Schwankungen bei den Prozessabschnitten 1 und 3 auftreten, also den Prozessabschnitten, in denen das Lager System involviert ist.

5.1.3 Diagramme und Berechnung des Korrelationskoeffizienten

Für die Betrachtung der Korrelation zwischen Durchlaufzeiten und Last ist auf der Y-Achse zusätzlich die gezählten Produktdaten als Counter aufgeführt worden.

Die helleren, durchgezogenen Linien der Durchlaufzeiten und der Counterdaten stellen die gesammelten Messpunkte dar. Die gepunkteten Linien stellen den gleitenden Mittelwert der fünften Periode dar. Diese sind zur besseren Übersicht des Verlaufs der Daten erstellt worden.

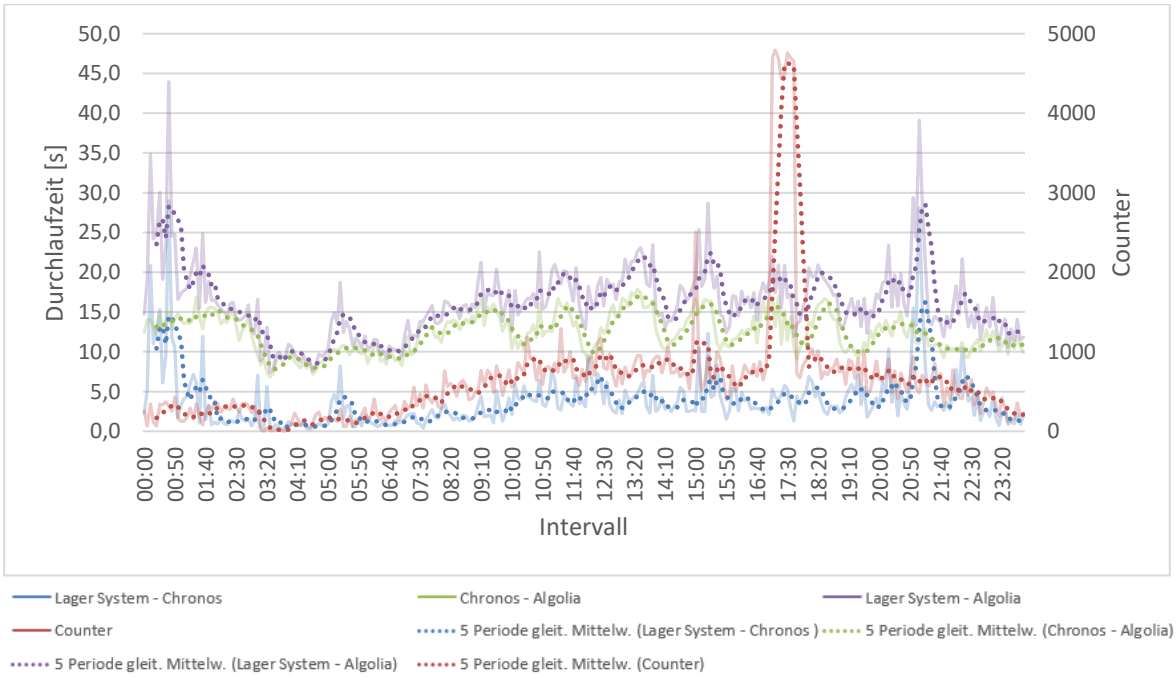


Diagramm 2. Durchlaufzeiten und der Counter

Im [Diagramm 2] sind die Durchlaufzeiten aus dem [Diagramm 1] zusammen mit den Counterdaten dargestellt. Dabei fällt der sehr starke Peak bei dem Counter im Zeitraum zwischen ca. 17 Uhr und ca. 18 Uhr auf. Das ist auf die täglichen Produktinformations-Updates zurückzuführen, die täglich in diesem Zeitraum durchgeführt werden.

Weiterhin ist zu sehen, dass die Durchlaufzeiten in diesem Zeitraum nicht genauso oder ähnlich stark ansteigen. Das liegt am Load-Balancer[9], der sich bei zu hoher Last einschaltet und die Last auf die vorhandenen Instanzen verteilt.

In der folgenden Tabelle sind die Korrelationskoeffizienten der drei Prozessabschnitte dargestellt.

	Prozessabschnitt 1	Prozessabschnitt 2	Prozessabschnitt 3
Korrelationskoeffizient	0,107	0,275	0,224

Tabelle 2. Darstellung der Korrelationskoeffizienten der einzelnen Prozessabschnitte

Die Bewertung, der nach Pearson [Brückler, 2018, S. 117 ff.] ermittelten Korrelationskoeffizient, erfolgt nach [Cohen, 1988, S.83], wie in Kapitel [2.6] näher erläutert ist.

Der Wertebereich des Korrelationskoeffizienten $(\boldsymbol{r})$ liegt zwischen den Werten -1 und +1. Dieser Betrag gibt die Stärke des Zusammenhangs zwischen den Variablen an.

- $\boldsymbol{r} = 0$ --> keine
- $\boldsymbol{r} = 0{,}1$ bis $\boldsymbol{r} = 0{,}3$ --> gering bis moderat
- $\boldsymbol{r} = 0{,}3$ bis $\boldsymbol{r} = 0{,}5$ --> moderat bis groß
- $\boldsymbol{r} > 0{,}5$ --> groß

Nach diesem Bewertungsschema ist eine geringe Korrelation im Prozessabschnitt 1 festzustellen, während in den Prozessabschnitten 2 und 3 eine leicht moderate Korrelation zu beobachten ist.

[9] Ein Load-Balancer ist ein Service, der sich um die Lastverteilung kümmert und bei einer zu großen Menge an Anfragen die Last auf mehrere parallellaufende Systeme verteilt.

5.1.4 Messergebnisse über eine Woche

Die nachfolgend aufgeführte [Tabelle 3] zeigt die berechneten Mittelwerte (MW) sowie das Minimum (Min) und Maximum (Max) der Durchlaufzeiten über die Kalenderwoche 11. Außerdem sind die Korrelationskoeffizienten (r) der Prozessabschnitte der Wochentage aufgeführt. Die letzte Spalte zeigt die über die Woche gemittelten Werte.

	14.3. Montag	15.3. Dienstag	16.3. Mittwoch	17.3. Donnerstag	18.3. Freitag	19.3. Samstag	20.3. Sonntag	**Gesamt**
Prozessabschnitt 1: Lager System – Chronos Service								
MW [s]	6,2	2,7	3,8	5,0	7,4	7,4	3,4	**5,1**
Min [s]	0,3	0,2	0,2	0,3	0,7	0,6	0,3	**0,2**
Max [s]	32,0	12,9	43,8	25,6	46,3	104,9	32,0	**104,9**
r	0,359	0,145	0,223	0,157	0,118	0,124	0,098	**0,175**
Prozessabschnitt 2: Chronos Service - Algolia								
MW [s]	12,2	12,9	12,7	12,6	12,7	12,4	10,4	**12,3**
Min [s]	6,7	8,1	8,1	7,5	8,6	7,7	7,4	**6,7**
Max [s]	18,8	16,7	16,7	16,4	17,6	17,1	16,3	**18,8**
r	0,261	0,310	0,298	0,241	0,158	0,190	0,243	**0,243**
Prozessabschnitt 3: Lager System - Algolia								
MW [s]	18,3	15,6	16,5	17,6	20,1	19,9	13,8	**17,4**
Min [s]	8,1	8,4	9,6	10,2	10,5	9,5	8,1	**8,1**
Max [s]	45,7	29,6	57,3	38,0	55,4	117,7	41,4	**117,7**
r	0,411	0,287	0,318	0,247	0,164	0,160	0,219	**0,258**

Tabelle 3. Darstellung der berechneten Werte einer Woche

Bei der Betrachtung der Durchlaufzeiten über eine Woche ist ein ähnliches Verhalten zu beobachten wie bei den Durchlaufzeiten eines Tages aus [vgl. Tabelle 1]. Die Prozessabschnitte 1 und 3, bei denen das Lager System involviert ist, zeigen über eine Woche betrachtet ebenfalls sehr starke Schwankungen. Diese liegen bei Prozessabschnitt 1 zwischen 0,2 Sekunden und 104,9 Sekunden und bei Prozessabschnitt 3 zwischen 8,1 Sekunden und 117,7 Sekunden.

Im Prozessabschnitt 2 sind diese starken Schwankungen der Durchlaufzeiten nicht zu beobachten, diese verlaufen im Bereich zwischen 6,7 Sekunden und 18,8 Sekunden.

Die über eine Woche ermittelten Korrelationskoeffizienten bewegen sich selbem Wertebereich, wie die eines Tages betrachteten aus [Tabelle 2]. Zu erwähnen ist hier, dass auch die Korrelationskoeffizienten mit unter starken Schwankungen unterliegen. Nichtsdestotrotz ist eine leichte Korrelation erkennbar.

Die Schwankungen zeigen, dass andere Einflussgrößen als die hier gemessenen, einen Einfluss auf die Durchlaufzeiten haben.

5.1.5 Durchlaufzeiten mit den drei größten gemessenen Schwankungen

Im nachfolgenden [Diagramm 3] sind die Durchlaufzeiten mit den drei größten gemessenen Schwankungen aus der [Tabelle 3] dargestellt, da Maximalwerte von über einer Minute beobachtet worden sind.

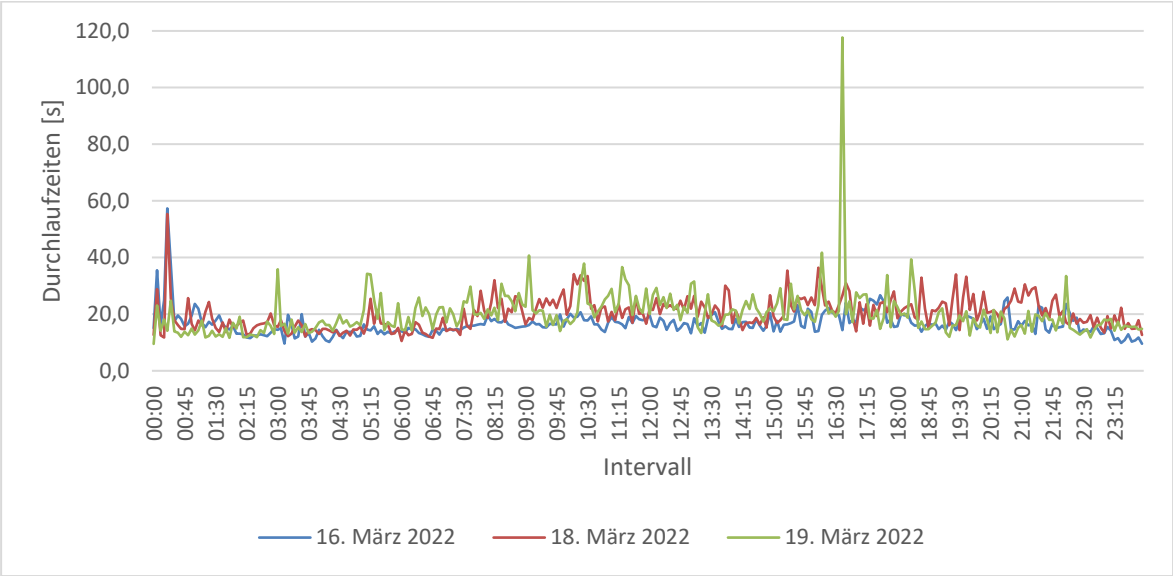


Diagramm 3. Durchlaufzeiten mit den drei größten gemessenen Schwankungen

Die Grafik zeigt, dass diese starken Peaks nur einmal am Tagen aufteten, also eine Seltenheit darstellen. Durch die geringe Häufigkeit dieser extremen Peaks werden die gemittelten Durchlaufzeiten nicht allzu stark beeinflusst.

5.2 Zusammenfassung und Diskussion der Ergebnisse

In diesem Abschnitt wird die Zusammenfassung der gesammelten Ergebnisse aus den Messungen aufgeführt und im Anschluss die daraus resultierenden Erkenntnisse diskutiert.

Die nachfolgende Tabelle fasst die Mittelwerte der Durchlaufzeiten, die berechneten Korrelationskoeffizienten sowie das Minimum und Maximum der einzelnen Prozessabschnitte über eine Woche zusammen.

	Prozessabschnitt 1	Prozessabschnitt 2	Prozessabschnitt 3
Mittelwert der Durchlaufzeiten [s]	5,1	12,3	17,4
Minimum [s]	0,4	7,7	9,2
Maximum [S]	42,5	17,1	55,0
Korrelationskoeffizient	0,175	0,243	0,258

Tabelle 4. Darstellung der berechneten Werte der einzelnen Prozessabschnitte

Die Durchlaufzeiten des überwachten Verarbeitungsprozesses liegen im Mittel (über eine Woche betrachtet) bei 17,4 Sekunden. Im Prozessabschnitt 1 und 3 zeigen die Durchlaufzeiten starke Schwankungen, was auf den Verarbeitungsprozess zwischen dem Lager System und Chronos Service zurückgeführt werden kann.

Die genaue Ursache dieser Schwankungen kann mit den entwickelten Microservice Chronos nicht genau analysierten werden, da sich das Lager System außerhalb der überwachten Service-Architektur befindet und deswegen keine Aussagen über dessen Verarbeitungsprozess getroffen werden kann. Durch diesen externen Effekt werden die Durchlaufzeiten für den Verarbeitungsprozess des Prozessabschnitts zwischen Lager System und Algolia anscheind mehr beeinflusst als die Last. Dies zeigt sich anhand der geringen Korrelation in Prozessabschnitt 1, mit dem Wert $(r = 0{,}175)$ im Vergleich zu einer gering moderaten Korrelation in Prozessabschnitt 2 mit dem Wert $(r = 0{,}243)$ und Prozessabschnitt 3 mit einem Wert $(r = 0{,}258)$.

Insgesamt kann ein geringer Zusammenhang zwischen der Durchlaufzeit und der Last ermittelt werden.

6 Abschließende Betrachtung

Mit dem entwickelten Microservice Chronos ist die Firma Bringmeister GmbH ein Tool bereitgestellt worden, um die Durchlaufzeiten über deren Verarbeitungsprozess ihrer Produktdaten zu messen und Analysen über den Zusammenhang zwischen Durchlaufzeit und Last durchführen zu können.

In den folgenden Abschnitten werden die gesammelten Erkenntnisse aus den vorherigen Kapiteln noch einmal zusammengefasst und daraus eine abschließende Betrachtung Abgeleitet. Weiterhin werden noch Ausblicke über mögliche Erweiterungen und Ansatzpunkte diskutiert, welche in einer weiterführenden Arbeit diesem Projekt hinzugefügt werden können.

6.1 Zusammenfassung und Fazit

Basis dieser Arbeit war die Erstellung eines Microservices, mit dem eine Überwachung des Verarbeitungsprozesses von Produktdaten durch die Service-Architektur ermöglicht wird.

In dem entwickelten Microservice wurde zunächst eine REST-API zur Aufnahme der XML-Dateien aus dem Lager System implementiert. Aus diesen XML-Dateien wird mit dem hier entwickelten Chronos Service der Zeitstempel als Startzeit für die Messung und die Produktdaten extrahiert. Mit diesen entnommenen Daten wird im Anschluss eine wiederholte Anfrage in eingestellten Intervallen an Algolia durchgeführt. Damit wird überprüft, ob das Produkt aktualisiert ist.

Nach Erhalt eines aktualisierten Produkts wird der aktuelle Zeitstempel gespeichert und als Endzeit der Messung verwendet. Damit ließ sich im Anschluss die Durchlaufzeit des Verarbeitungsprozesses berechnen. Mittels einer Zählung von Events, konnte die Last ermittelt werden, die während der Messung auf dem System lastet. Der Chronos Service ist so entwickelt worden, dass die Messergebnisse sich über eine REST-API als CSV-Datei herunterladen lassen.

Mit dem Chronos Service sollte ermittelt werden, wie die durchschnittlichen Durchlaufzeiten des Verabeitungsprozess von Produktdaten sind und ob ein Zusammenhang zwischen der Durchlaufzeit und der Last des Systems existiert.

Über eine Woche betrachtet liegt die Durchlaufzeit im zeitlichen Mittel bei 17,4 Sekunden, wobei ein Miximum von 117,7 Sekunden beobachtet wurde. Aus Sicht der Firma Bringmeister GmbH ist die durchschnittliche Durchlaufzeit ein akzeptabler Wert, da die Produktdaten bei einem Verarbeitungsprozess zwei externe Systeme sowie mehrere interne Services und zwei Eventqueues durchlaufen. Auch das Maximum ist für die Firma

Bringmeister GmbH ein hinnehmbarer Wert, da solche Ausreißer im untersuchten Zeitraum nur jeweils einmal täglich auftreten.

Da die Durchlaufzeiten starke Schwankungen aufweisen, wurde ein zusätzlicher Messpunkt eingebaut, um die Ursache besser eingrenzen zu können. Betrachtet man die Durchlaufzeiten der beiden Teilstrecken (Lager System - Chronos Service und Chronos Service - Aloglia), ließ sich beobachten, dass die Ursache dieser Schwankungen auf das Lager System zurückzuführen ist, da die Schwankungen nur bei den Teilstrecken auftreten, bei denen das Lager System beteiligt ist.

Die Ursache dieser Schwankungen kann mit den entwickelten Microservice Chronos nicht genau ermittelt werden, da das Lager System außerhalb der Systemgrenzen des Fachbereichs Connect-Backend liegt.

Die Durchlaufzeiten steigen bei großer Last nicht so stark an, was durch einen Load-Balancer abgefedert wird. Dieser schaltete sich bei hoher Last ein und verteilte diese auf die Instanzen, in dem die gesamte System-Architektur des Fachbereichs Connect-Backend läuft.

Mit der Berechnung des Korrelationskoeffizienten nach Person [Brückler, 2018, S. 117 ff.] konnte herausgefunden werden, dass ein geringer Zusammenhang zwischen der Durchlaufzeit und der Last des überwachten Verarbeitungsprozesses besteht.

6.2 Grenzen und Kritik

In technischen Systemen ist eine exakte Messung von Zeiten schwer realisierbar, da jedes System für sich selbst Last erzeugt, wodurch die Messergebnisse verfälscht werden. Zum Messen der Last des Chronos Services können Analysenverfahren angewandt werden. Mit diesen lässt sich die Verarbeitungszeit eines Systems ermitteln, die im Anschluss aus den gemessenen Zeiten rausgerechnet werden können.

Die Ursache der teils starken Variationen der Durchlaufzeiten zwischen dem Lager System und dem Chronos Service kann mit dem hier entwickelten Service nicht genau analysiert werden, da das Lager System nicht zur Service-Architektur des Fachbereichs Connect-Backend gehört.

Anhand der gesammelten Erkenntnisse zeigt sich, dass andere Einflussgrößen, als die hier diskutierte Last, einen Einfluss auf die Durchlaufzeiten haben. Welche Einflussgrößen hierbei eine Rolle spielen, lässt sich mit den hier eingesetzten Methodiken nicht ermitteln.

Mit der Berechnung des Korrelationskoeffizienten lässt sich zwar eine hinreichende Aussage über den Zusammenhang zwischen der Durchlaufzeit und der Last treffen, diese sagt aber nichts darüber aus, ob auch tatsächlich eine Kausalität zwischen diesen beiden

Werten existiert. Damit eine genauere Aussage darüber getroffen werden kann, müssten weitere Analysen und Berechnungen durchgeführt werden. Für diese Arbeit war eine quantitative/hinreichende Betrachtung ausreichend.

6.3 Ausblick

Der implementierte Chronos Service ist ein eigenständiger und funktionsfähiger Microservice, welcher aber einige Verbesserungs- und Erweiterungsmöglichkeiten bietet, die im Folgenden aufgeführt werden.

- Die Firma Bringmeister GmbH wird in naher Zukunft Algolia durch einen anderen Service ersetzen. Der Chronos Service ist so entwickelt worden, dass er flexibel erweiterbar ist, sodass der neue Service einfach integriert werden kann. Auf diese Weise lassen sich die Durchlaufzeiten des Verarbeitungsprozesses für den neuen Service ermitteln.

- In dieser Arbeit sind nur Produktinformationen der Verfügbarkeitsdaten analysiert worden. Weitere denkbare Erweiterungsmöglichkeiten sind die Analyse von Preis - und Produktinformationsdaten. Dafür muss der gesamte Verarbeitungsprozess für diese Produktdaten erweitert werden, der in diesen Microservice entwickelt worden ist.

- Es könnten weitere Messpunkte für den Verarbeitungsprozess eingebunden werden, um zusätzliche Verarbeitungszeiten zwischen den einzelnen Services zu ermittelten. Des Weiteren können auch die Verarbeitungszeiten der einzelnen Services selbst ermittelt werden, wodurch noch genauere Analysen der Durchlaufzeiten möglich wären. Zusätzlich könnten die Durchlaufzeiten des Verarbeitungsprozesses für die umgekehrte Richtung analysiert werden, womit eine Aussage über die Verarbeitungszeit des gesamten Verarbeitungsprozesses (Hin -und Rückweg) von Produktdaten getroffen werden kann.

7 Literaturverzeichnis

Adams, P., et al. (2012), „SOAP“, verfügbar unter https://www.w3.org/TR/soapjms/ (Zugriff am 12. Januar 2022).

Apache (2021), „CSV“, verfügbar unter https://commons.apache.org/proper/commons-csv/user-guide.html#Example:_Parsing_an_Excel_CSV_File (Zugriff am 3. Februar 2022).

Bittmann, F. (2016), „Einführung in die moderne Kausalanalyse“, verfügbar unter http://felix-bittmann.de/index.php/artikel/19-einfuehrung-in-die-moderne-kausalanalyse?showall=1 (Zugriff am 16. März 2022).

Brückler, F. M. (2018), *Geschichte der Mathematik: Das Wichtigste aus Analysis, Wahrscheinlichkeitstheorie, angewandter Mathematik, Topologie und Mengenlehre*, XII, 1. Aufl., Springer-Verlag GmbH, Deutschand.
ISBN: 978-3-662-55573-6

Buschmann, F. (Hg.) (1996), *Pattern-Oriented: Software Architecture: A System of Patterns*, Bd. 1., John Wiley & Sons, Chichester.
ISBN: 978-0-471-95869-7

Cohen, J. (1988), *Statistical Power Analysis for the Behavioral Sciences,* 2. Aufl., Erlbaum, New York.
ISBN: 978-0-8058-0283-2

Gamma, E. (Hg.) (2015), *Design Patterns: Entwurfsmuster als Elemente wiederverwendbarer objektorientierter Software,* 1. Aufl., mitp Verlags GmbH & Co.KG, Frechen.
ISBN: 978-3-8266-9700-5

Haleby, J. (2010), „Awaitility“, verfügbar unter URL https://github.com/awaitility/awaitility/wiki/Usage (Zugriff am 29. Januar 2022).

Humboldt-Universität (2019), „Bravais-Pearson-Korrelationskoeffizient“, verfügbar unter https://wikis.hu-berlin.de/mmstat/Bravais%E2%80%93Pearson%E2%80%93Korrelationskoeffizient (Zugriff am 23 Febraur 2022).

Kluever, K. A. (2021), „Guava“, verfügbar unter https://github.com/google/guava/wiki/CachesExplained (Zugriff am 12. Februar 2022).

Mockito (2021), „Unit-Test“, verfügbar unter https://site.mockito.org/ (Zugriff am 13. Dezember 2021).

Oracle (2016), „Java Documentation. JAXB“, verfügbar unter https://docs.oracle.com/javase/8/docs/technotes/guides/xml/jaxb/index.html (Zugriff am 11. Dezember 2021).

Raymond, E. S. (2003), *The Art of Unix Programming,* 1. Aufl., Prentice Hall, Boston. ISBN: 978-0131429017

Richardson, C. (2014), „What are microservices?“, verfügbar unter https://microservices.io/ (Zugriff am 5. Januar 2022).

Schnell, G. & Hoyer, Konrad (1986), *Mikrocomputer: Interface Fibel,* 2. Aufl., Friedr. Vieweg & SohnVerlagsgesellschaft mbH, Deutschland. ISBN: 9783528142483

Spring (2002), „Async“, verfügbar unter https://spring.io/ (Zugriff am 8. Dezember 2021).

Srivastava, S. (2020), „Differences Between ZonedDateTime and OffsetDateTime“, verfügbar unter https://www.baeldung.com/java-zoneddatetime-offsetdatetime (Zugriff am 10. Dezember 2021).

TenMedia (2022), „Monitoring“, verfügbar unter https://www.tenmedia.de/de/glossar/monitoring (Zugriff am 16. Dezember 2021).

Ullenboom, C. (2021), „Java ist auch eine Insel“, verfügbar unter https://openbook.rheinwerk-verlag.de/javainsel/ (Zugriff am 16. Dezember 2021).

Voigt, K.-I. (2022), „Durchlaufzeit“, verfügbar unter https://wirtschaftslexikon.gabler.de/definition/durchlaufzeit-32490 (Zugriff am 3. Februar 2022).