OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

HSP-Studienarbeit

Erstellung eines Datentransformations- und Verteilungssystems für SaaS-Anwendungenl

eingereicht von:	Stephan Nunhofer
	Matrikelnummer: 3247646
	Studiengang: Master Informatik
	OTH Regensburg
betreut durch:	Prof. Dr. Johannes Schildgen
	OTH Regensburg

Kallmünz, der 14. Juli 2020

Inhaltsverzeichnis

1 Aktuelle Bedeutung der SaaS-Anwendungen

Mit *Software-as-a-Service* (SaaS) Anwendungen werden Programme bezeichnet, welche dem Kunden als Dienstleistung angeboten wird, jedoch auf der IT-Infrastruktur des Dienstleisters betrieben wird. Der Kunde kann also auf den Dienst zugreifen, ohne eine eigenen ausreichende Umgebung für dessen Betrieb zu besitzen. Meist erfolgt dieser Zugriff über Web-Schnittstellen [1]. Durch die für den Kunden einfache Nutzung steigt der Bedarf nach der derartigen Angeboten.

So prognostiziert das Beratungsunternehmen Gartner, dass Produkte auf Basis von SaaS auch in den folgenden zwei Jahren deutlich mehr Umsatz generieren werden, wie in Abbildung 1.1 zu sehen ist.

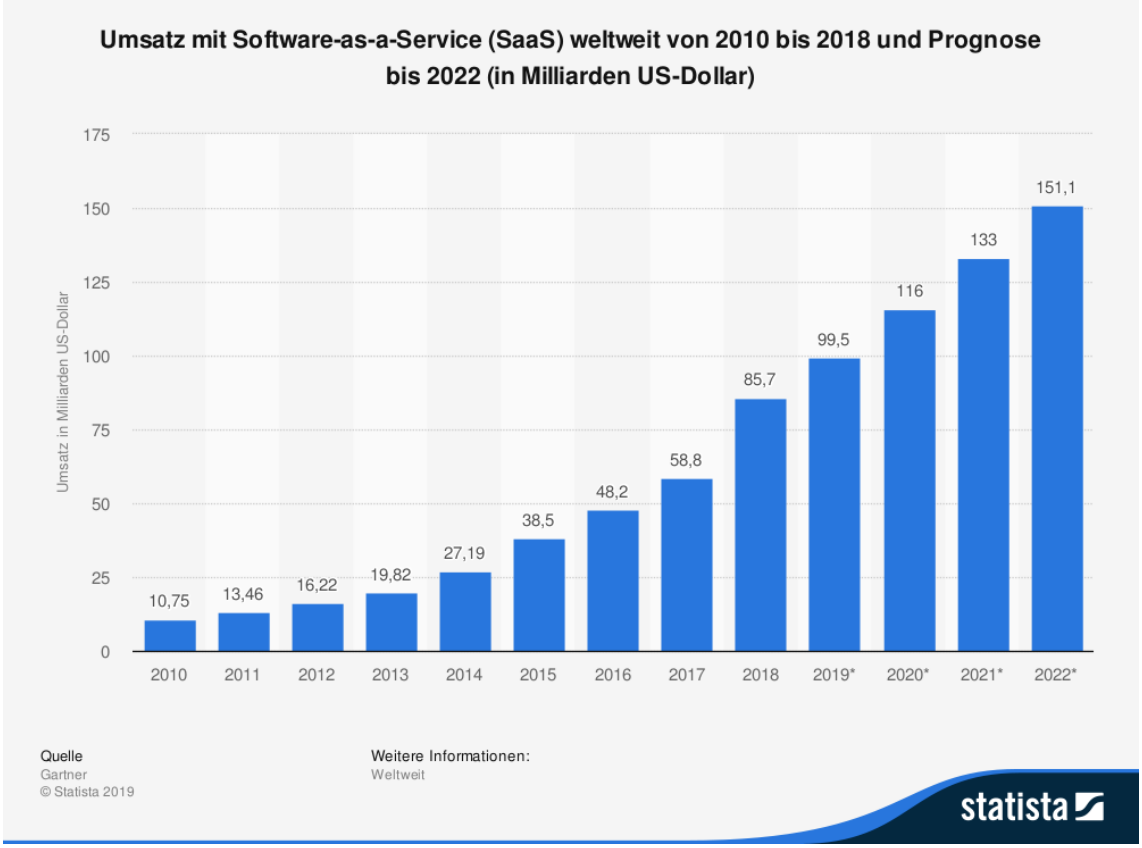

Abbildung 1.1: Prognose zum Umsatz mit *Software-as-a-Service* Weltweit bis 2022 [2]

SaaS-Anwendungen sind zudem der stärkste Umsatzzweig nach Gartner den Abstand zu

den *Infrastructure-as-a-Service* (IaaS) Anwendungen ausbauen, wie in Abbildung 1.2 zu sehen ist [3].

	2018	**2019**	**2020**	**2021**	**2022**
Cloud Business Process Services (BPaaS)	41.7	43.7	46.9	50.2	53.8
Cloud Application Infrastructure Services (PaaS)	26.4	32.2	39.7	48.3	58.0
Cloud Application Services (SaaS)	85.7	99.5	116.0	133.0	151.1
Cloud Management and Security Services	10.5	12.0	13.8	15.7	17.6
Cloud System Infrastructure Services (IaaS)	32.4	40.3	50.0	61.3	74.1
Total Market	**196.7**	**227.8**	**266.4**	**308.5**	**354.6**

Abbildung 1.2: Weltweite Prognose des Umsatzes mit öffentlichen *Cloud*-Diensten [3]

Da allerdings eine Vielzahl an Diensten vorhanden ist, sind die Daten oft zwischen diesen verteilt. Möchte nun ein Kunde das Angebot wechseln und stellt der neue Anbieter keine Konvertierungsmöglichkeit für die Daten des anderen Anbieters bereit, so muss der Kunde entweder die Daten selbstständig übertragen oder auf diese verzichten. Dieses Problem kann jedoch durch Extrahierung in Injektion der Informationen durch die gegebenen Anwendungsschnittstellen und eine externe Konvertierung in einem neuen Anwendungsprogramm behoben werden. Für die Erstellung eine Prototypen wird ein Projekt mit Simon Hofmeister und Stephan Nunhofer unter der Aufsicht von Prof. Dr. Johannes Schildgen gestartet.

2 Umsetzung des Datentransformations- und Verteilungssystems

Die Hauptaufgaben dieses Werkzeuges ist die Gewinnung der Daten aus den verschiedenen Diensten. Diese Daten werden dann in einer Datenbank gespeichert und können bei einer Abfrage zur Übertragung in einen anderen Dienst verändert werden. Zur Ermöglichung dieses Ziels müssen die Unterschiedlichen Informationen aus den Anwendungen auf eine gemeinsame Datenmenge zusammengeführt werden. Darauf folgt die Festlegung der Klassenstruktur, eine Definition des Speichervorganges in die Datenbank und die Umsetzung der Rückführung in den selben oder einen anderen Dienst. Das Projekt fokussiert sich auf Dienstleister in den Bereichen Kalender- und Notiz-Verwaltung, wobei zu jedem Bereich jeweils drei Angebote eingebunden werden. Die Notiz-Verwaltung, auf welche sich diese Arbeit hauptsächlich konzentriert, wird dabei von *Keep* (Google), *OneNote* (Microsoft) und *Notion* (Notion) vertreten. Zur Umsetzung wird die Programmiersprache *Python* genutzt und die Verbindung mit den Diensten über entsprechende *Wrapper-Klassen* in dieser Sprache realisiert. Als Datenbank wird nur das Datenbanksystem *MongoDB* genutzt.

2.1 Zusammenführung der Daten

Als erster Schritt werden die Daten aus den verschiedenen Schnittstellen aufgelistet. Dafür werden alle möglichen Attribute, welche man aus den Diensten extrahieren kann, in einer Liste gesammelt und diese mit den Eigenschaften der anderen Dienste verglichen. Gibt es Übereinstimmungen in Bezeichnung oder Funktion, wir dieses Attribut für das gemeinsame Datenformat akzeptiert. Unter den gemeinsamen Daten gibt es dabei rein informative Informationen, wie beispielsweise den Titel oder einen Text, sowie für die Funktion des Dienstes strukturell notwendige Werte, wie die ID. Diese sind zwar für jede Anwendung vorhanden, unterscheiden sich allerdings nahezu jedes mal in ihren Werten. Bei der Verwendung dieser als gemeinsames Attribut muss noch eine Anpassung der Werte vorgenommen werden. Der Vorgang ist für die Notiz-Anwendungen in Abbildung 2.1 zu sehen.

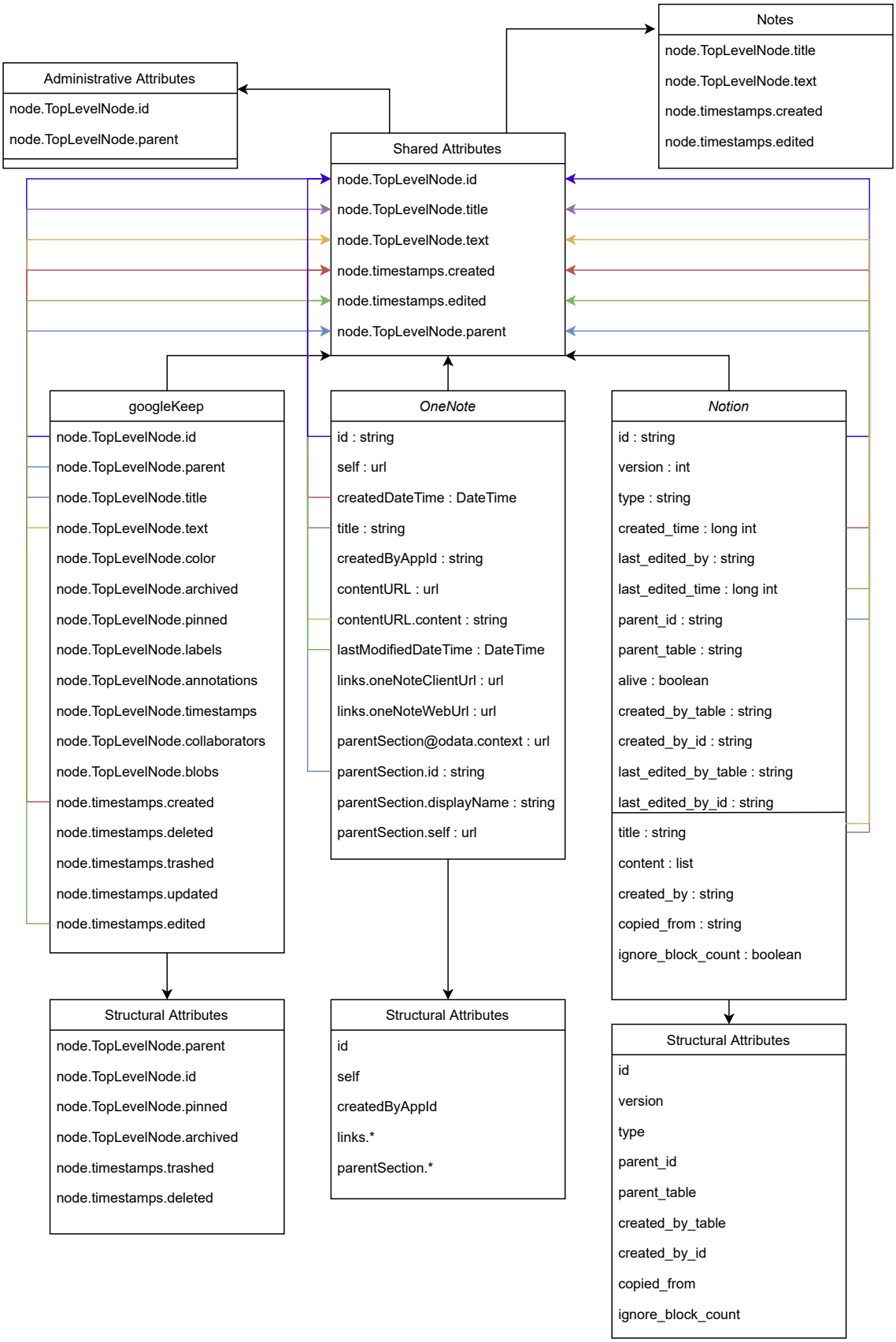

Abbildung 2.1: Darstellung des Attribut-Vereinigungsprozesses

Nach der Vereinigung dieser Attribute mit denen aus den Kalender-Diensten, wurden folgende Attribute in Tabelle 2.1 für als feste, zwingend notwendige Eigenschaften eines internen Datensatzes definiert.

Tabelle 2.1: Gemeinsame Attribute aller Dienste

Wert	**Datentyp**	**besondere Formatierung**
id	string	<Diensttyp>#<Dienstklassenname>#<Dienst-ID>
title	string	
text	string	
created	string	<J>-<M>-<T>T<S>:<M>:<S>.<ZZ>Z (UTC-Zeitformat)
edited	string	<J>-<M>-<T>T<S>:<M>:<S>.<ZZ>Z (UTC-Zeitformat)

Jeder Datensatz aus den Anwendungen muss diese Attribute enthalten, um in die Datenbank aufgenommen zu werden. Die übrigen Eigenschaften, welche nicht in allen anderen Diensten enthalten sind, werden ebenfalls in die Datenbank übertragen. Bei diesen muss jedoch bei der Rückführung in den Dienst die Existenz geprüft werden.

2.2 Festlegung der Klassenstruktur

Um sicherzustellen, dass die festen Attribute vorhanden sind, werden die Daten in einer Klasse statt einer Datenstruktur gespeichert. Dabei gibt es eine Basisklasse, welche die festen Attribute enthält, und mehrere abgeleitete Klassen mit den diensteigenen, optionalen Werten. Diese Klassen besitzten keine Methoden, da sie als reine Speicherklassen genutzt werden. Objekte dieser Klassen werden durch die Dienst-Schnittstellenklassen erstellt. Ihre Aufgabe ist die Kommunikation mit den Diensten über deren jeweilige Benutzerschnittstellen. Sie übernehmen also die Anmeldung, die Extrahierung von Daten aus der Anwendung und die Injektion der Daten aus der Datenbank zurück in den Dienst. Sie fungieren außerdem als Bedienungsschnittstelle für die Nutzeroberfläche. Dabei werden auch hier alle Dienst-Schnittstellenklassen von einer Basis-Klasse abgeleitet, wodurch man sicherstellt, dass alle Unterklassen auf die gleichen Resourcen zugreifen und die gleiche Funktionalität bereitstellen. Die letzte Klassengruppe stellen die Datenbank-Schnittstellenklassen dar. Diese ermöglichen den Datentransfer zu und von der Datenbank. Ebenso wie bei den Dienst-Klassen, wird auch hier eine Vererbungsstruktur genutzt, um eine Konsistenz zwischen den abgeleiteten Klassen sicherzustellen. Das gesammte Konzept ist dabei auf hohe Flexibilität ausgelegt. So kann ein neuer Dienst einfach über das Hinzufügen einer neuen Dienst-Klasse eingebunden und die Datenbank über eine neue Datenbank-Klasse gewechselt werden. In beiden Fällen ist entweder keine oder nur wenige Zeilen an Änderungen in den vorhanden Klassen nötig. Durch das vereinheitlichte Datenkonzept ist zudem eine hohe Kompatibilität gegeben. Eine Übersicht über die Klassenstruktur ist in Abbildung 2.2 zu sehen. Die Dienst-Klassen sind dabei die *apiInterfaces*, die Datenbank-Klassen die *datastores* und die Speicherklassen die *dataObjects*.

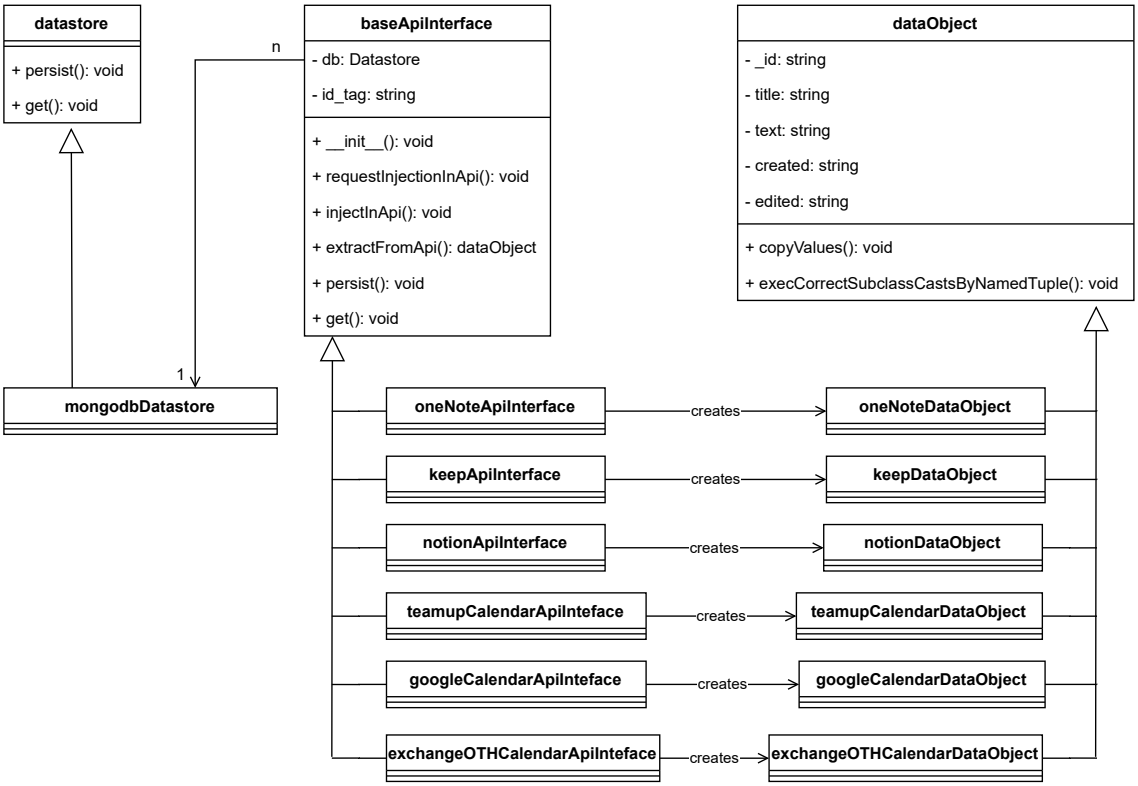

Abbildung 2.2: Vereinfachtes Klassendiagramm zur Darstellung der Klassenstruktur

2.3 Umsetzung des Speichervorganges

Um ein Element aus der Anwendung zu Extrahieren muss dessen Benutzerschnittstelle angesprochen werden. Zuerst meldet sich die Schnittstellenklasse bei dem Dienst an und erhält ein Zugriffsobjekt. Bei den Notizdiensten wird mit diesem Objekt wird dann zuerst eine Suche nach allen vorhandenen Elementen gestartet und deren Attribute in die jeweilige Speicherklasse übertragen. Dabei wird die *_id* mit einer zusätzlichen Zeichenkette aus dem Diensttypen und der Ursprungsklasse versehen. Dies ist dann beispielsweise der Zusatz *notes#keepApiInterface#* für den *Keep-Dienst* des Unternehmens *Google*. Zudem werden alle Zeitangaben auf das in Tabelle 2.1 dargestellte Zeitformat konvertiert. Die Speicherung der Daten-Klasse übernimmt die Methode *persist*. Diese nutzt das Attribut *db* aus der Basisklasse, um die *persist*-Methode aus der Datenbank-Klasse aufzurufen. Im Prototyp der *mongodbDatastore*. Dieser wiederum stellt eine Verbindung zur Datenbank her und erhält dessen Verbindungsobjekt. Die Speicherklasse wird in eine *dict*-Speicherstrucktur umgewandelt und in den Dienst eingefügt. Es wird also eine Sammlung von Schlüssel-Werte-Paaren übergeben. Die Konvertierung ist dabei im Falle des *mongodbDatastores* nötig, da die Verbindungsklasse keine Klassen als Eingabeparameter akzeptiert und eine Konvertierung in ein *dict* leicht zu realisieren ist. Die Injektion in die Datenbank findet mithilfe der *replace_one()*-Methode von *MongoDB* statt. Dabei ist die *upsert*-Flagge gesetzt, um sicherzustellen, dass nicht vorhandene Einträge eingefügt werden. Existieren diese bereits, ersetzt die Methode sie durch den neuen, übergebenen Eintrag, was einer Aktualisierung gleichkommt.

2.4 Definition des Extraktionsvorganges aus der Datenbank und der Datenrückführung in einen Dienst

Der Rückweg aus der Datenbank in die Dienste beginnt dabei ebenfalls in den Schnittstellenklassen. Die Notiz-Klassen bereiten in der Methode *requestInjection* die Ergebniszähler vor und rufen die Methode *requestInjectionInApi* auf. Dessen Parameter stammen dabei aus den übergebenen Werten der aufrufenden Methode. Diese haben dabei auch Standartwerte, welche dem Nutzer eine selektierte Angabe ermöglichen. Die Parameter sind in Tabelle 2.2 mit ihrem Datentypen und ihrer Bedeutung aufgelistet.

Tabelle 2.2: Parameter für die Methoden zur Rückholung der Daten aus der Datenbank

Parameter	**Datentyp**	**Bedeutung**
substrIdTag	string	Stellt die Teilzeichenkette dar, nach welcher die _id-Felder in der Datenbank gefiltert werden sollen.
filterOptions	Liste aus dicts	Speichert die zusätzlichen Filteroptionen für die MongoDB Aggregationsmethode.
transformationOptions	Liste aus dicts	Enthält die Datentransformationsinformationen für die MongoDB Aggregationsmethode. Beispielsweise werden Werte geändert oder Variablenwerte anderen Variablen zugewiesen.
addAggOptions	Liste aus dicts	Damit können weitere MongoDB Parameter für die Aggregationsmethode angegeben werden. Beispielweise ist durch eine Angabe des *$unset*-Befehls eine Löschung eines Feldes möglich.

Die Methode *requestInjectionInApi* aus der Basis-Klasse leitet diese wiederum an die *get*-Methode der Datenbank-Klasse weiter. Jedoch kommt hier der Parameter *serviceObject* hinzu, welcher einfach eine Referenz auf das aufrufende Objekt - also die Schnittstellenklasse - ist. Die aufgerufene Methode des *mongodbDatastore* stellt dann zuerst eine Verbindung zur Datenbank her und definiert aus den übergebenen Werten eine *aggregation pipeline*. Damit werden die Daten nach den angegebenen Optionen gefiltert, transformiert oder anders verändert und an die Datenbank-Klasse als *dict* zurückgegeben. Für jeden Eintrag in dieser Wertesammlung wird nun die *injectInAPI*-Methode der aufrufenden Schnittstellen-Klasse über die übergebene Referenz aufgerufen. Diese meldet sich bei dem jeweiligen Dienst an und überprüft zuerst, ob die *id* aus dem Eintrag schon im Dienst vorhanden ist. Ist das der Fall, werden die Einträge aktualisiert. Wenn nicht, wird dieser als neuer Eintrag eingefügt. Die Nebeninformationen, welche in den abgeleiteten Klassen von *dataObject* gespeichert werden, müssen dabei auf Existenz überprüft werden, da die Einträge auch von anderen Diensten stammen können. Eine weitere Aufgabe ist die Registrierung und das Abfangen von Fehlern bei der Injektion. Entsteht ein Fehler wird dieser abgefangen, ausgeben und

die Variable *errorCount* inkrementiert. Kann der Eintrag problemlos verarbeit werden, erhöht sich die Variable *successCount*. In der Klasse *oneNoteApiInterface* ergibt sich dabei eine Besonderheit. Da der Dienst nur eingeschränkte Änderungen an den Einträgen zulässt, wird bei erkennen eines Eintrages im Dienst die letzte Änderungszeit überprüft. Ist der Eintrag im Dienst jünger als der in der Datenbank wird die Variable *ignoredCount* inkrementiert und der Eintrag übersprungen. Andernfalls wird dieser neu eingefügt und als Erfolg gezählt. Nachdem alle Einträge eingefügt oder aktualisiert wurden, gibt die Ursprungsmethode *requestInjection* noch die Variablen *errorCount*, *ignoredCount* und *successCount* als Ergebnis aus. In Abbildung 2.3 ist die Ausgabe auf der Konsole für den gesamten Injektonsprozess zu sehen.

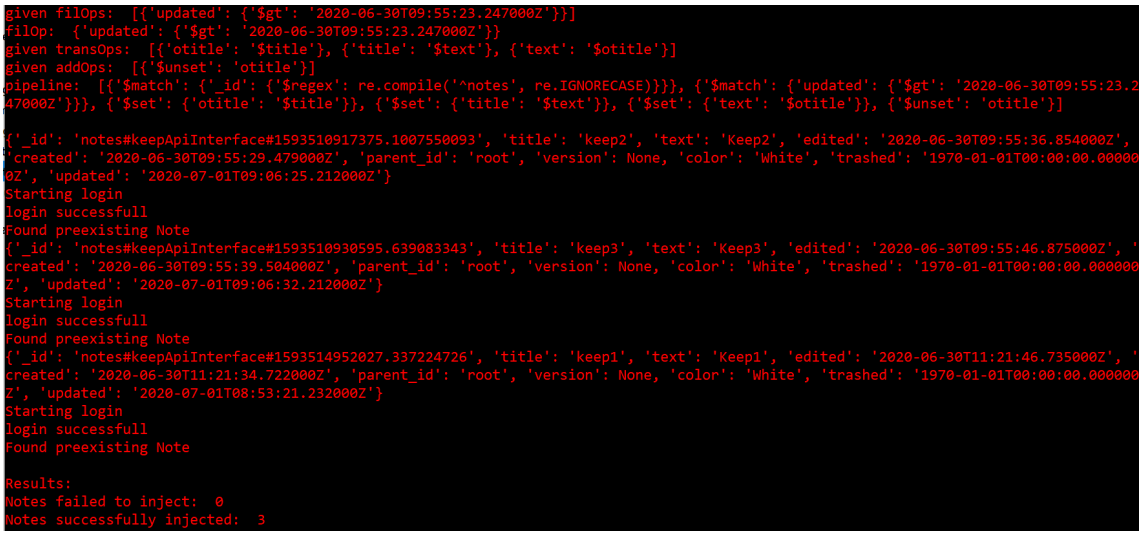

Abbildung 2.3: Konsolenausgabe für den Injektionsvorgang

2.5 Aufbau der graphischen Oberfläche

3 Mögliche zukünftige Anwendungen des Datentransformations- und Verteilungssystems

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] McNee, W: *SaaS 2.0.* Journal of Digital Asset Management, 3:209–214, August 2007. https://link.springer.com/article/10.1057/palgrave.dam.3650088#citeas.

[2] Gartner: *Umsatz mit Software-as-a-Service (SaaS) weltweit von 2010 bis 2018 und Prognose bis 2022 (in Milliarden US-Dollar)*, Juli 2020. https://de.statista.com/statistik/daten/studie/194117/umfrage/umsatz-mit-software-as-a-service-weltweit-seit-2010/.

[3] *Gartner Forecasts Worldwide Public Cloud Revenue to Grow 172020*, November 2019. https://www.gartner.com/en/newsroom/press-releases/2019-11-13-gartner-forecasts-worldwide-public-cloud-revenue-to-grow-17-percent-in-2020.