APPE FS2020 Entwicklung eines Bestellsystems

PROJEKTDOKUMENTATION GRUPPE 03

Gabriela Moos, Frederico Fischer, Oliver Werlen

Hochschule Luzern,

Versionskontrolle:

Rev.	Datum	Autor	Bemerkungen	Status
1.0	28.05	alle	Finale Anpassungen	final
0.6	27.05	Oliver Werlen Frederico Fischer Gabriela Moos	Klassendiagramme und Beschreibung Order, Filiallager, Reorder Beschreibung Schnittstellen Beschreibung Kundenservice	
0.5	09.04	Federico Fischer Oliver Werlen Gabriela Moos	Beschreibuung manueller Testfall hinzugefügt Architektur Diagramm aktualisiert	
0.4	03.04.2020	Federico Fischer Gabriela Moos	Erster Entwurf Microservice Diagramm Mappingtabelle Microservices/Projekte auf Gitlab	
0.3	25.03.2020	Federico Fischer Oliver Werlen Gabriela Moos	- Kontext-Diagramm - Liste der Szenarien Prozessanalyse - User Stories - Bounded context diagram	
0.2	07.03.2020	Federico Fischer Gabriela Moos	- Entwurf Kontext Diagramm - Entwurf Liste der Prozesse/Szenarien - Epics	
0.1	26.02.2020	Gabriela Moos	- Dokumentstruktur - Entwurf Zeitplan	

Inhalt

1. Projektmanagement

1.1. Projektverantwortlichkeiten

Name	**Aufgaben**
Frederico Fischer	Scrum Master
Gabriela Moos	Scrum Team Member
Oliver Werlen	Scrum Team Member
Jörg Hofstetter	Product Owner
Roland Christen Roland Gisler Jörg Hostetter	Auftraggeber

Tabelle 1: Verantwortlichkeiten im Projektteam 3

1.2. Zeitplanung

Zwischenziel für Sprint 2

Als Team haben wir uns folgende Zwischenziele für nach dem Sprint 2 vorgenommen:

- Wir haben eine grobe Architekturübersicht über alle geplanten Microservices und deren Zusammenspiel aufgezeichnet.
- Wir haben einzelne Microservices entwickelt und getestet, welche einzelne Prozesse aus dem Epic Nr. 1 abdecken.
- Wir verstehen das Zusammenspiel der einzelnen Microservices und dem Messagingsystem auf technischer Ebene und können weitergeplante Prozesse effizienter im System implementieren.

Bei den Epic- und Prozessnummern referenzieren wir uns auf die in diesem Dokument definierten Nummern.

Projektstart: SW 01

Projektabschluss: SW15

Iterationen: 4 Sprints: S1, S2, S3, S4

Phase	Meilensteine	Woche	Artefakte/Deliverables/Termine
Initialisierung	Projektstart	SW01 20.2.	Teambildung
		SW02 27.2.	
		SW03 5.3.	Techbootcamp, Arbeitsumgebung eingerichtet, GitLab-Projekte vorhanden
Konzeption	Fachreview, Umsetzung S1	SW04 12.3.	
		SW05 19.3.	
		SW06 26.3.	Fachreview • Kontextdiagramm • Szenarioliste • Prozessanalyse • Domainmodell • Zeitplan mit Ziel für nach Sprint 2, • Stories/Epics für Sprint 1 auf GitLab Minimal Techreview • Softwareumgebung läuft gemäss Tech.Bootcamp
Realisierung	Sprintreview S1	SW07 2.4.	• Software auf VCS, läuft auf Integrationsumgebung (nicht lokal!) • Demo der einzelnen Stories • Aktuelles Architekturdiagramm Stories bereinigt (was ist erledigt? Was nicht? Neue Stories) • Vorbereiten für kurze Reflexion: • Planung Sprint 2
	S2 Sprintreview S2	SW08 9.4.	Architektur Review: Architekturmodell 10.4.: Karfreitag
		SW09 16.4.	
		SW10 23.4.	Sprintziel S2 • Epic Nr. 1 abgeschlossen mit Ausnahme von den Prozessen 13 und 9 (Mahnung, Rechnung, Bestellbestätigung) • Epic Nr. 6 abgeschlossen Sprintreview S2 • Software auf VCS, läuft auf Integrationsumgebung (nicht lokal!)

Phase	Meilensteine	Woche	Artefakte/Deliverables/Termine
			• Demo der einzelnen Stories • Aktuelles Architekturdiagramm Stories bereinigt (was ist erledigt? Was nicht? Neue Stories) • Vorbereiten für kurze Reflexion • Planung Sprint 3
	S3, Sprintreview S3	SW11 30.4.	
		SW12 7.5.	Sprintreview S3 • Software auf VCS, läuft auf Integrationsumgebung (nicht lokal!) • Demo der einzelnen Stories • Aktuelles Architekturdiagramm Stories bereinigt (was ist erledigt? Was nicht? Neue Stories) • Vorbereiten für kurze Reflexion • Planung Sprint 4
	S4, Sprintreview S4	SW13 14.5.	
		SW14 21.5.	Do, 21.5.: Auffahrt. Sprintreview S4 • Software auf VCS, läuft auf Integrationsumgebung (nicht lokal!) • Demo der einzelnen Stories • Aktuelles Architekturdiagramm Stories bereinigt (was ist erledigt? Was nicht? Neue Stories) • Vorbereiten für kurze Reflexion
Abschluss	Projektabschluss	SW15 28.5.	Projektabschluss

Tabelle 2: Projektrahmenplan Team 3

2. Kontext-Diagramm

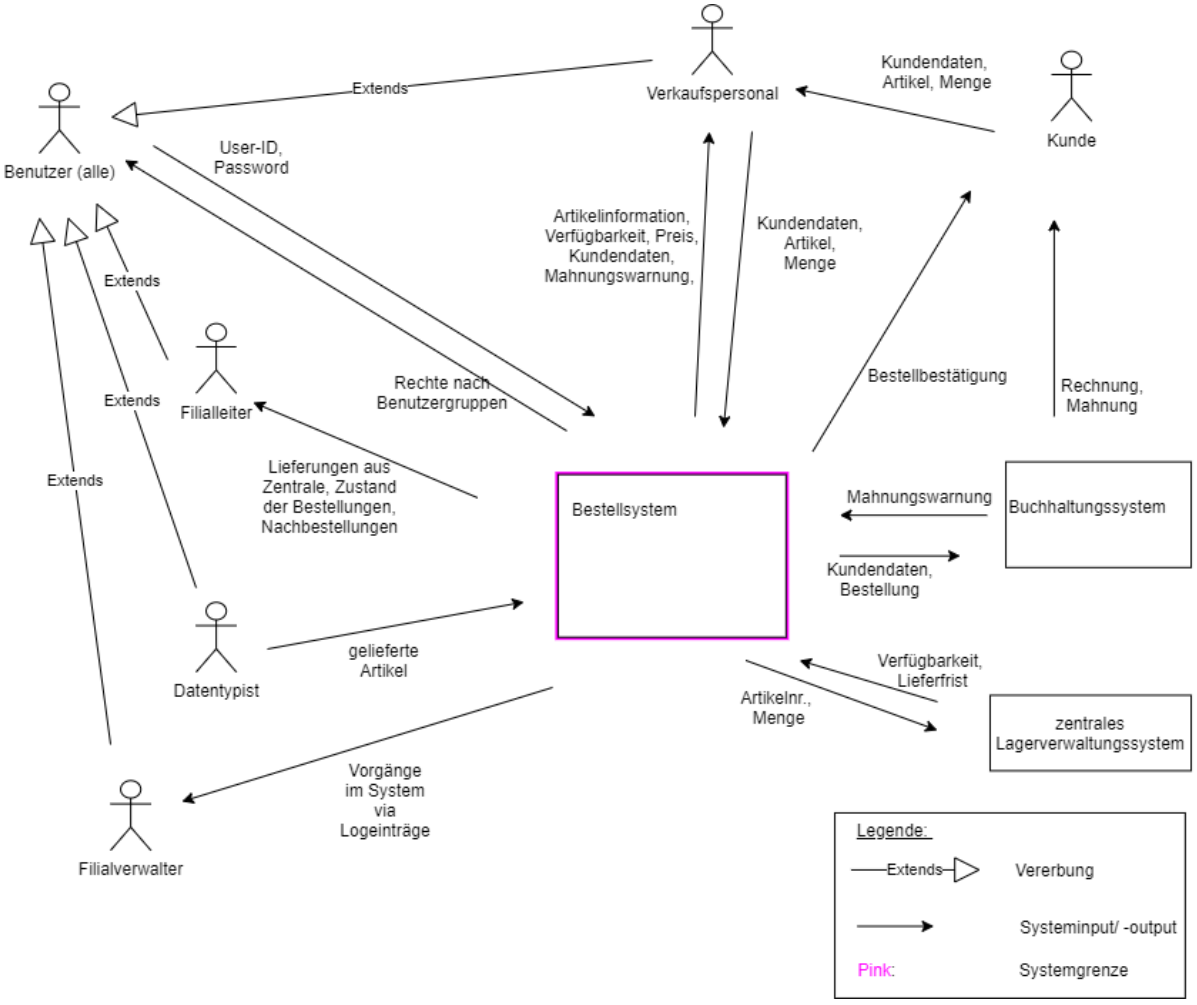


Abbildung 1: Kontextdiagramm Bestellsystem

3. Liste der wichtigen Prozesse/ Szenarien

Nr	Prozess/Szenario	Akteur
1.	Gelieferte Artikel im Filiallager hinzufügen / eintragen.	Datentypist/in
2.	Sortiment der Filiale definieren.	Filialleiter/in
3.	Minimalen Lagerbestand eines Artikels festlegen	Filialleiter/in
4.	Wird die Artikelmenge im Filiallager bezüglich der Minimalmenge unterschritten, wird dieser Artikel automatisch aus dem Zentrallager nachbestellt.	Bestellsystem
5.	Benutzer erfassen und Rollen zuweisen: SysAdmin, Filialleiter/in, Verkäufer/in, Datentypist/in	Filialverwalter/in = SysAdmin
6.	Vorgänge im System anhand von Logs aufzeichnen.	Filialverwalter/in
7.	Akteure loggen sich im Bestellsystem ein.	Datentypist/in, Filialleiter/in, Filialverwalter/in, Verkaufspersonal
8.	Eine Bestellung im System erfassen, wobei der Artikel ausgewählt, die Verfügbarkeit geprüft und die Bestellung ausgeführt wird.	Verkaufspersonal
9.	Wird eine Bestellung ausgeführt, wird automatisch eine Rechnung erzeugt und der Kunde erhält eine Bestellbestätigung.	Bestellsystem
10.	Änderungen an der Bestellung vornehmen.	Verkaufspersonal / Filialleiter/in
11.	Eine Bestellung annullieren.	Verkaufspersonal / Filialleiter/in
12.	Alle vorhandenen Artikel in einer Übersichtsliste anzeigen.	Verkaufspersonal / Filialleiter/in
13.	Der Akteur / die Akteurin erhält eine Warnung, wenn es zu einer ausstehenden Mahnung durch einen Kunden kommt.	Verkaufspersonal
14.	Alle aktuellen Bestellungen einsehen.	Verkaufspersonal / Filialleiter/in
15.	Alle aktuellen Nachbestellungen einsehen.	Verkaufspersonal / Filialleiter/in
16.	Alle Lieferungen einsehen.	Filialleiter/in

Tabelle 3: Liste der wichtigen Prozesse/Szenarien

4. Einfache Prozessanalyse für «Bestellung ausführen»

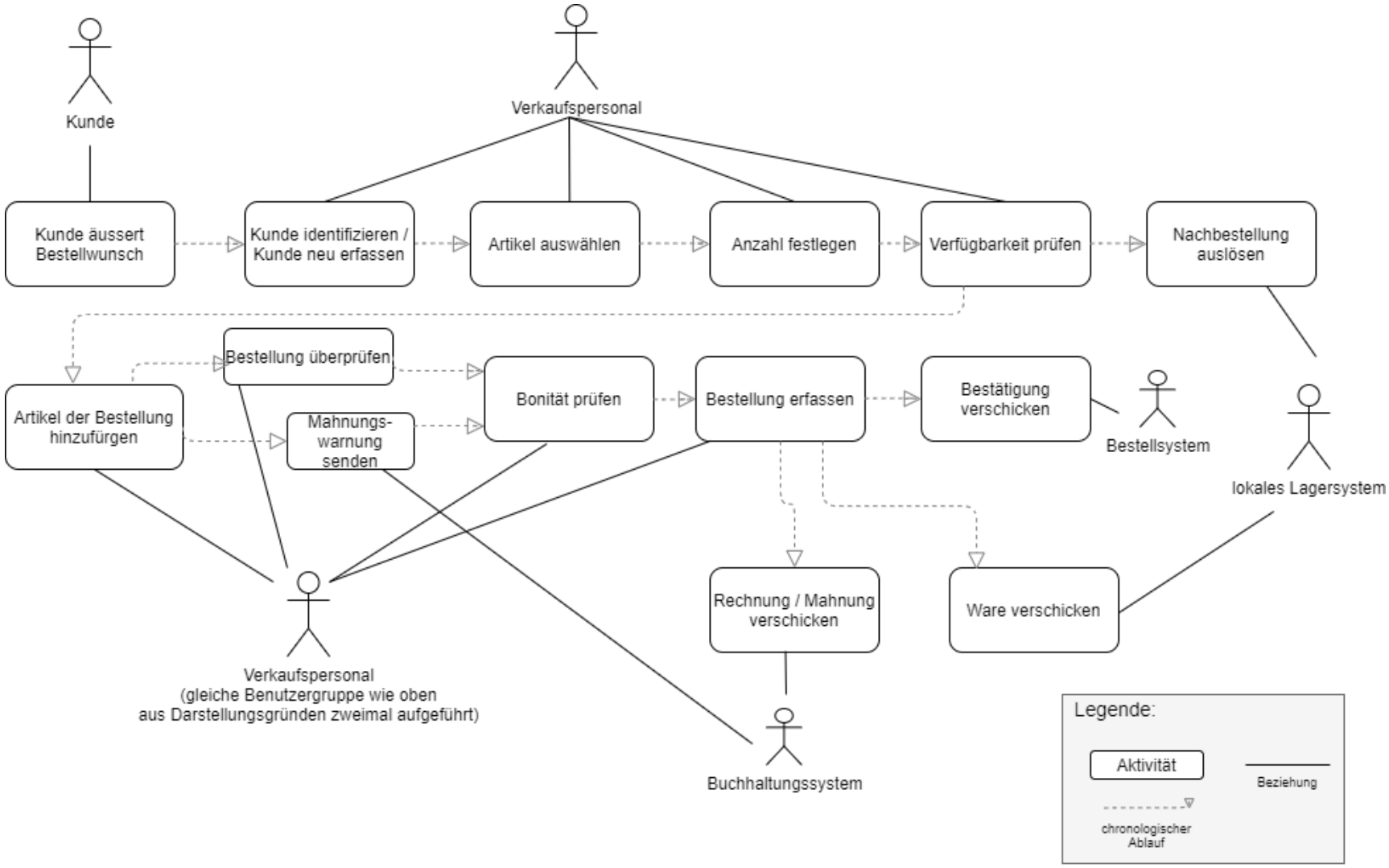


Abbildung 2: Prozessanalyse für "Bestellung ausführen"

5. Domänen-Modell

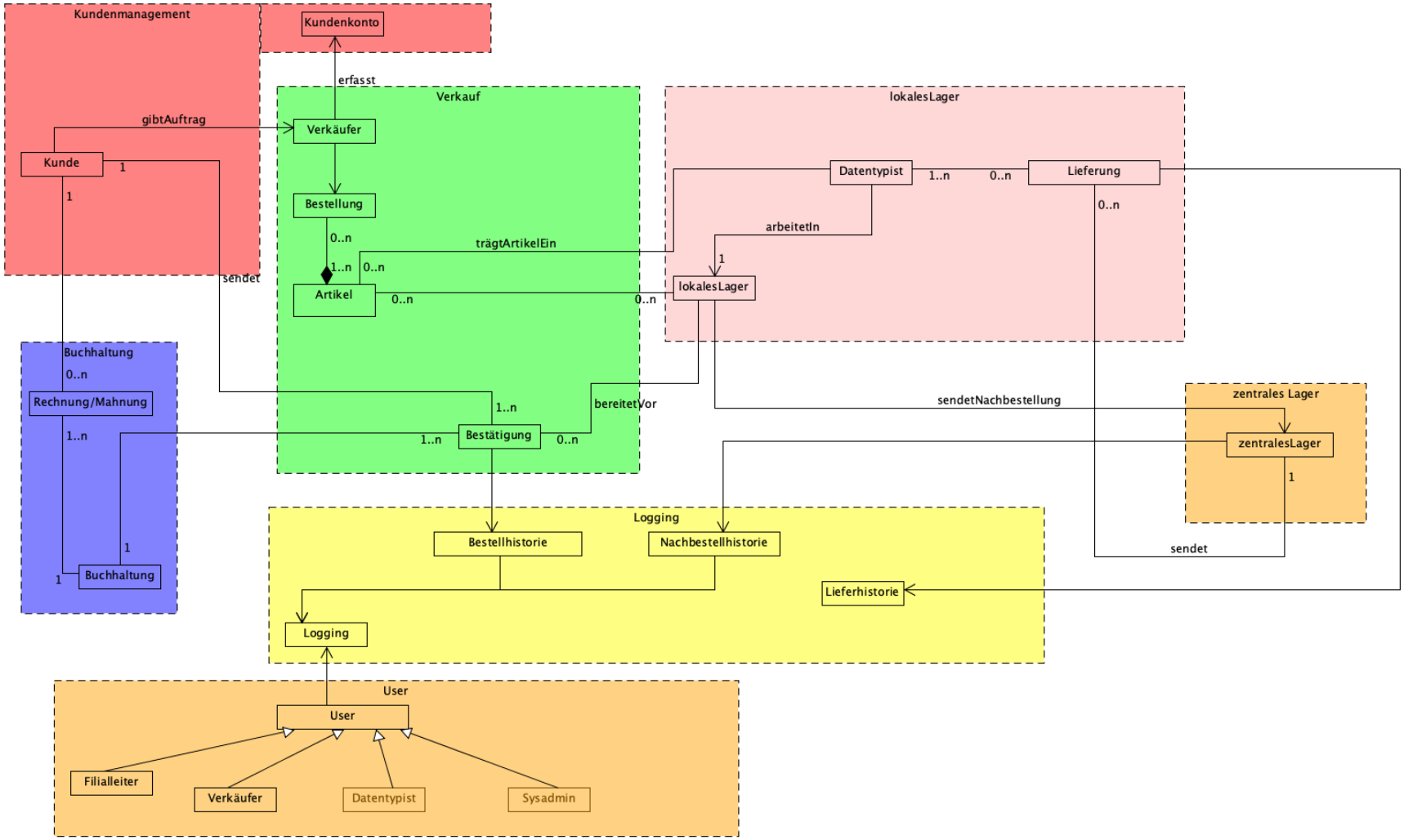


Abbildung 3: Domänenmodell zum Bestellsystem

6. Microservice-Modell

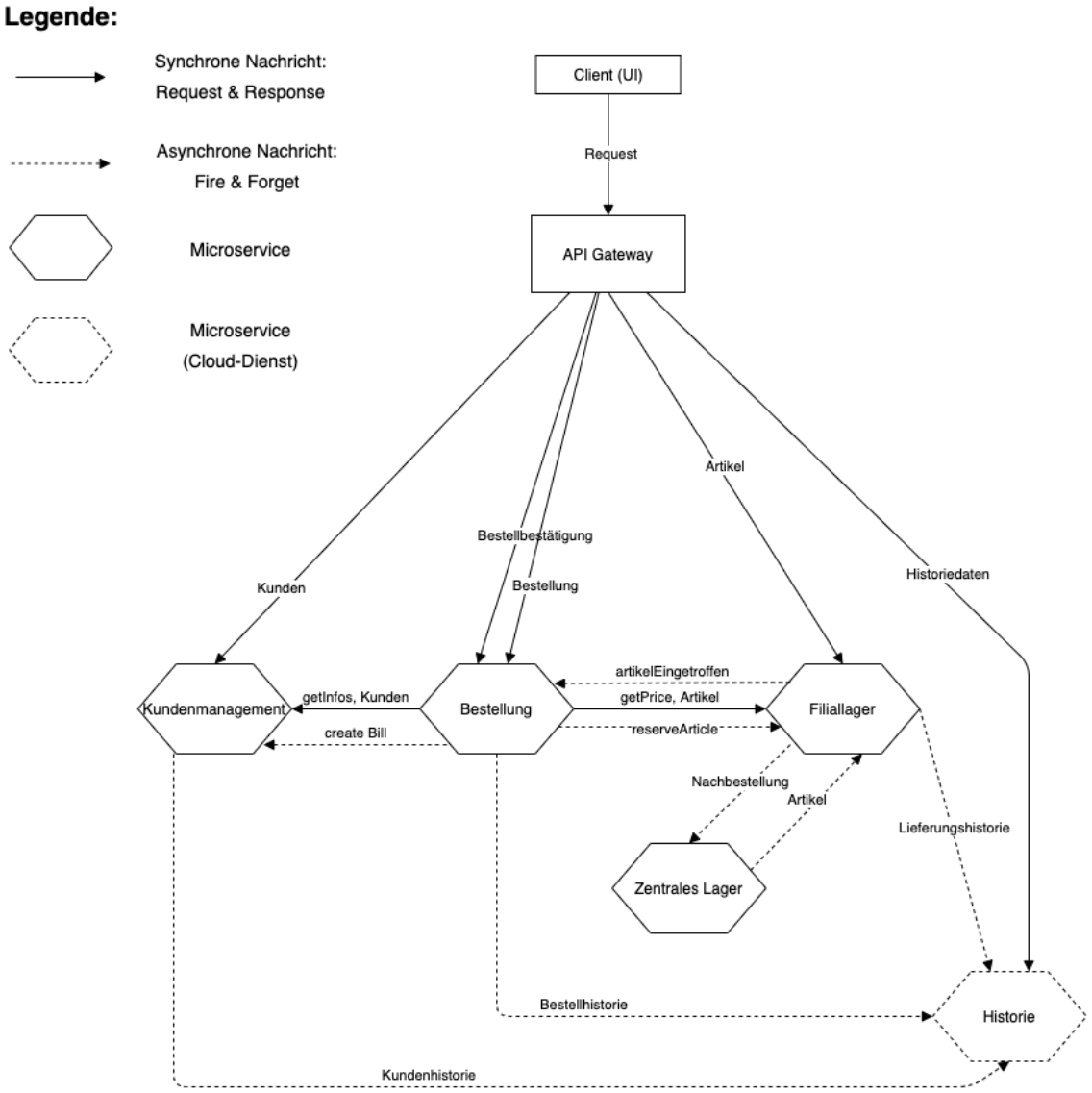


Abbildung 4: : Architektur der Microservices

Microservice	**Projektname auf Gitlab**
Kundenmanagement	g03-customermanagement
Bestellmanagement	g03-ordermanagement
Filiallager	g03-localwarehouse_new
Historie	LogZ.io
Gateway	g03-RESTinterface

Tabelle 4: Übersicht Microservices und zugehörige Projekte auf Gitlab

7. Detailbeschreibung der Microservices

7.1. Kundenmanagement

7.1.1. Einführung

Der Service Kundenmanagement verwaltet die Kundendaten. Die Kundendaten werden auf dem Frontend vom Verkaufspersonal eingegeben und gelangen via synchrone Anfrage vom Restinterface an den Kundenservice.

7.1.2. Beteiligte Prozesse

Das Kundenmanagement ist an folgenden Prozessen beteiligt:

- Kunde suchen
- Kunde erfassen
- Bestellung ausführen (vgl. Kapitel 7.2.2)
- Rechnung erstellen (vgl. Rechnung erstellen in Kapitel 7.2.2)
- Bonität (Mahnungslevel) prüfen (vgl. Bestellung ausführen in Kapitel 7.2.2)

In diesem Abschnitt werden nur die Prozesse Kunde suchen und Kunde erfassen beschrieben. An den anderen Prozessen ist der Kundenservice indirekt beteiligt via Bestellservice. Sie sind beim Bestellservice beschrieben.

Kunde suchen: Wenn ein Kunde etwas bestellen möchte, sucht das Verkaufspersonal zuerst nach dem Kunden via Name oder via KundenID. Es ist auch möglich, sich alle Kunden anzeigen zu lassen. Wird der Kunde gefunden, liefert der Kundenservice den Kunden an die aufrufende Anwendung.
Kunde erfassen: Wenn ein Kunde nicht gefunden wurde, erfasst das Verkaufpersonal die Kundendaten. Wenn keine KundenID mitgelifert wird, wird ein neuer Kunde erstellt und der Kunde wird zusammen mit einer neu generierten KundenID zurückgeschickt. Falls bei der Erfassung eine gültige ID mitgeliefert wird, wird der existierende Datensatz mit den neuen Informationen aktualisiert. Wird eine ungültige ID mitgeliefert, wird die Anfrage

zurückgewiesen mit dem Hinweis, dass die ID ungültig ist.

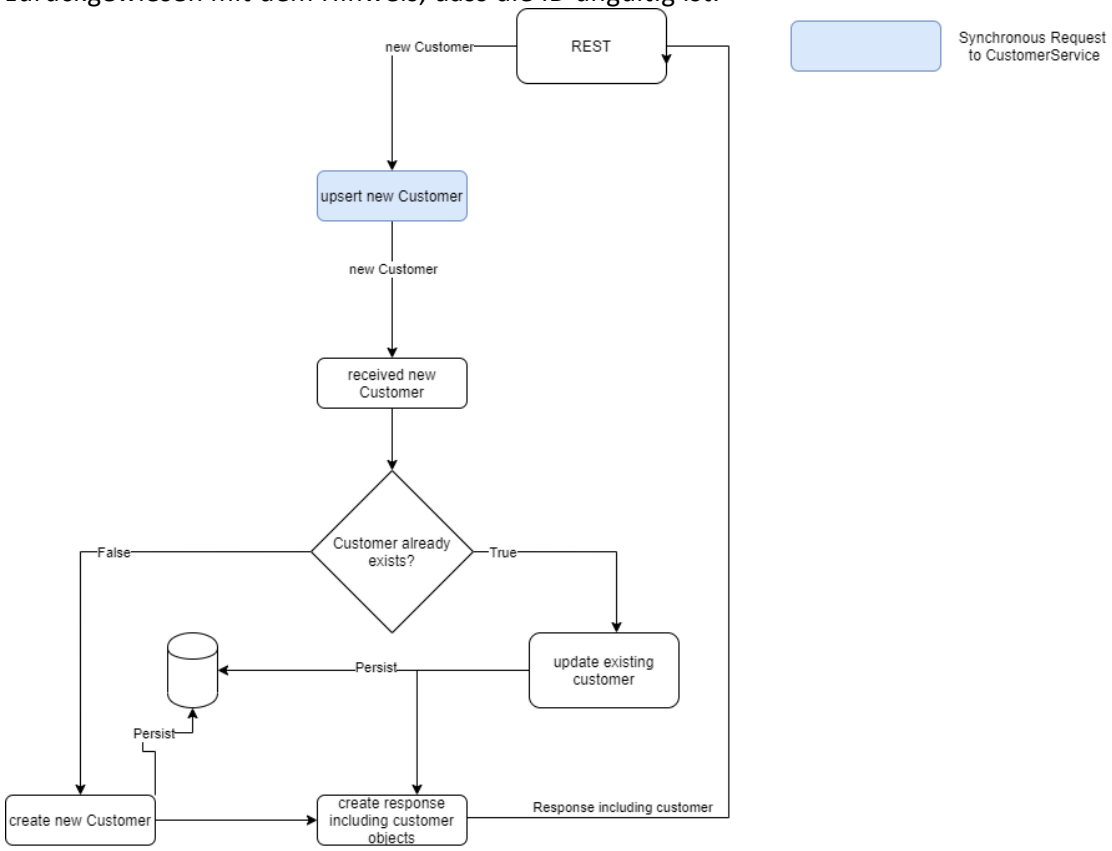


Abbildung 5: Prozess Kunde erfassen

7.1.3. Daten

Beim Kundenmanagement wird alles gehandhabt, was den Kunden angeht. Das beinhaltet auch den Mahnstatus und die Rechnung. Darauf werden folgende Einheiten verwaltet:

Kunde	- KundenID - Vorname - Nachname - Strassenname - Postleitzahl - Ort - Email - Telefonnummer (Optional) - Mahnungslevel
Rechnung	- RechnungsID - Datum der Rechnungserstellung - BestellungsID - KundenID - Vorname des Kunden - Nachname des Kunden - Artikelliste - Preis (Rechnungstotal)

Tabelle 5: Einheiten des Kundenmanagements

7.1.4. Aufbau

Anhand des folgenden vereinfachten Klassendiagramms soll aufgezeigt werden, wie das Kundenmanagement aufgebaut ist:

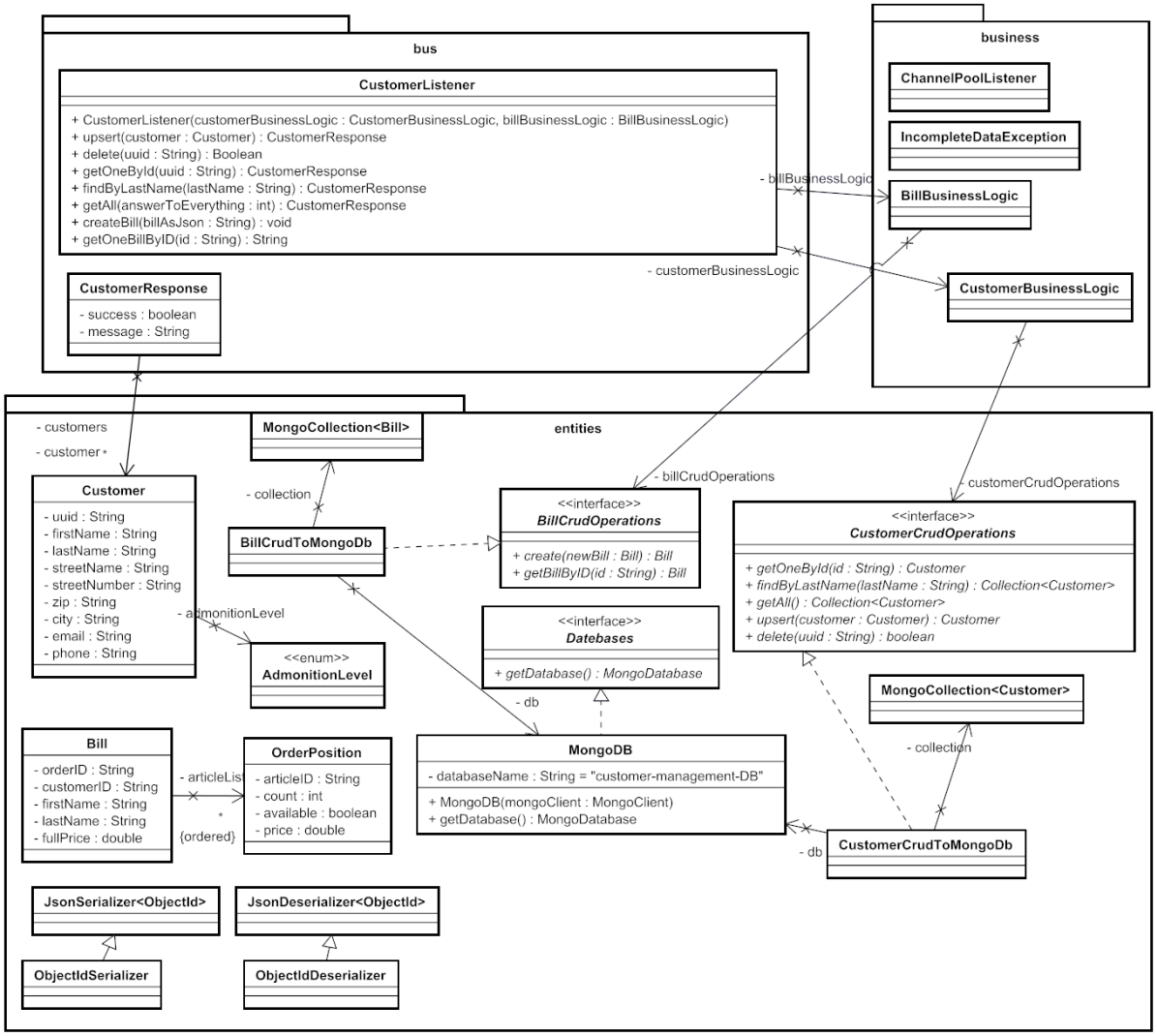


Abbildung 6: Klassendiagramm des Kundenservices

7.1.4.1. Package «micro»

Application: In Abbildung 6 aus Platzgründen nicht abgebildet ist das package micro. Es beinhaltet die Klasse Application mit der main methode, mit der Micronaut gestartet wird.

7.1.4.2. Package «bus»

CustomerListener: hört auf die Queues des Kundenservices.

CustomerResponse: wird an den Service gesendet, der den Customer Service aufruft. Neben dem boolean success, das meldet, ob die Anfrage erfolgreich war, wird eine message mitgeliefert. Wenn ein Kunde gesucht oder geändert wurde, liefert die Message neben dem Messagetext auch die Kunden.

7.1.4.3. Package «business»

ChannelPoolListener: erstellt die Queues des Kundenservices und bindet sie an den Exchange g03 im RabbitMQ. Die dazugehörigen Routing Keys sind im Service g03-restinterface in der Klasse CustomerClientSync interface definiert.

IncompleteDataException: wird von Customer BusinessLogic geworfen, wenn Pflichtfelder nicht ausgefüllt sind. Die Exception enthält eine Liste mit Namen der Felder, in denen Daten fehlen.

CustomerBusinessLogic: überprüft, ob alle Pflichtfelder ausgefüllt sind. Sammelt die Felder mit fehlenden Informationen in einer Liste und gibt diese an die IncompleteDataException weiter. Prüft, ob die Postleitzahl 4-stellig ist.

BillBusinessLogic: bekommt vom Aufrufer die Informationen zur Bestellung und der KundenID. Holt sich vom Kunden Name und Vorname des Kunden und erstellt zusammen mit der Bestellinformation eine Rechnung.

7.1.4.4. Package «entities»

Customer: enthält die Kundeninformation. Wird als Pojo verwendet, das an den aufrufenden Service geschickt wird. Die Kunden werden in der Datenbank persitiert.

AdmonitionLevel: Enum mit den Mahnstufen nothing, first level, second level, third level.

Bill: enthält die Rechnungsinformation. Wird in der Datenbank gespeichert.

OrderPosition: enthält Art, Anzahl, Preis und Verfügbarkeit des bestellten Artikels. Wird in der Bestellung verwendet.

Databases: Interface mit der Methode getDataBase, die eine MongoDatabase liefert.

MongoDB: implementiert das Interface Database. Erstellt einen MongoClient und eine Datenbank vom Typ MongoDatabase.

MongoCollection<Bill>: Interface für eine MongoCollection, die Bill Objekte aufnimmt.

MongoCollection<Customer>: Interface für eine MongoCollection, die Customer Objekte aufnimmt.

CustomerCrudOperations: Interface für die CRUD Operationen der Kundendaten auf der Datenbank. Enthält die Methoden upsert (insert new, update existing), getOneById, findByLastName, delete. Delete wird nur intern zum Testen verwendet, es wird nicht nach aussen an andere Services angeboten.

CustomerCrudToMongoDB: implementiert CustomerCrudOperations und MongoCollection<Customer> und handelt das eigentliche Einfügen und Holen der Kunden aus der Datenbank.

BillCrudOperations: Interface für die CRUD Operationen der Rechnungsdaten auf der Datenbank. Enthält die Methoden create und getBillByID.

BillCrudToMongoDB: implementiert BillCrudOperations und MongoCollection<Bill> und handelt das eigentliche Einfügen und Holen der Rechnung aus der Datenbank.

ObjectIdSerializer/ObjectIdDeserializer: serialisieren, deserialisieren die in MongoDB verwendete ObjectID, so dass die ID als String vorliegt**.**

7.2. Bestellmanagement

7.2.1. Einführung

Das Bestellmanagement entstand im Domänenmodel durch den Verkauf bzw. deren Tätigkeiten. Deshalb setzt sich vor allem der Verkäufer mit diesem Service auseinander. Dieser ist auch der einzige Service, welcher synchrone Anfragen tätigt.

7.2.2. Beteiligte Prozesse

Das Bestellmanagement ist an folgenden Prozessen beteiligt:

- Bestellung ausführen
- Artikelverfügbarkeit prüfen
- Artikelverfügbarkeit geprüft
- Rechnung erstellen
- Datenbankabfragen

Bestellung ausführen: In diesem Prozess kommt das Bestellmanagement zum Zug, wenn eine Anfrage, um eine Bestellung zu erfassen, versendet wird. Dabei werden die Kunden-ID als auch eine Liste mit Artikel-IDs an den Bestellservice synchron zugestellt. Anschliessend wird der Kunde nach dessen Mahnungslevel überprüft. Wenn dieser eine Mahnung hat, wird die Bestellung abgebrochen und der Verkäufer wird entsprechend benachrichtigt. Ansonsten wird das Filiallager mit der Artikelliste synchron benachrichtigt. Dabei wird der Preis der Artikel angefragt. Wenn die Antwort erfolgreich zurückkommt, wird die Bestellbestätigung erstellt und an den Verkäufer zugestellt. Anschliessend wird eine asynchrone Nachricht mit der Artikelliste und der Bestell-ID an das Filiallager zugestellt welches im Prozess «Artikelverfügbarkeit prüfen» erläutert wird.

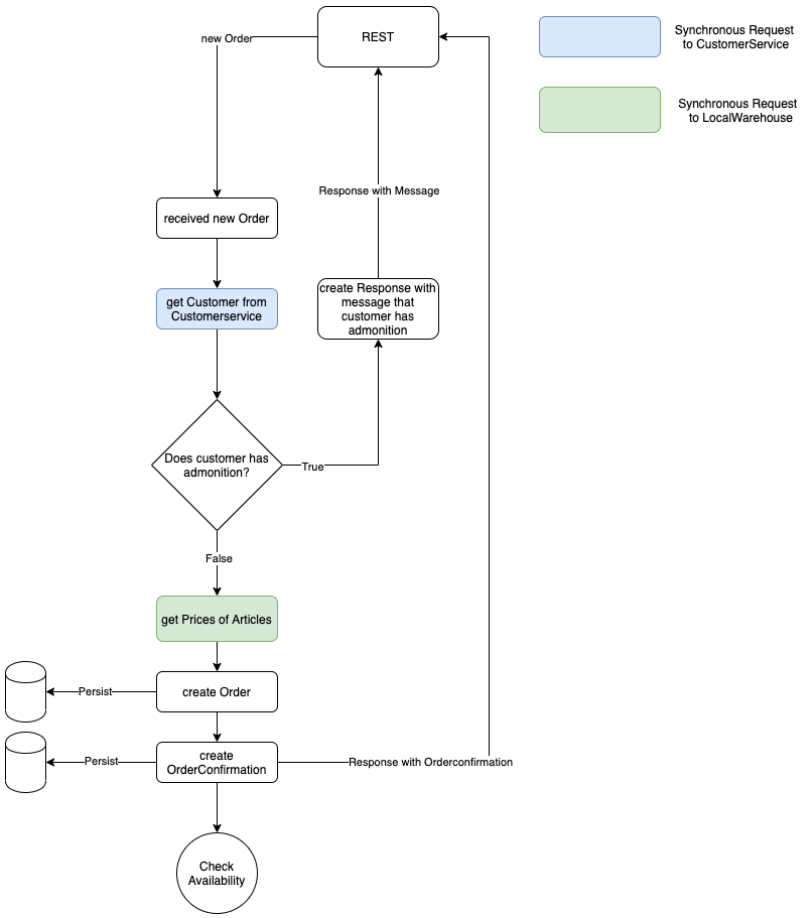


Abbildung 7: Prozess "Bestellung ausführen"

Artikelverfügbarkeit prüfen: Dieser Prozess wird nach der Erstellung der Bestellung ausgeführt. Dabei wird ein asynchroner Aufruf vom Bestellmanagement an das Filiallager gemacht, wobei die bestellten Artikel nach deren Verfügbarkeit überprüft werden sollen.

Artikelverfügbarkeit geprüft: Hierbei wird auf der Queue «Order | order.availabilityResponse» auf Messages gewartet, welche durch den Prozess «Artikelverfügbarkeit prüfen» angestossen wurden. Dabei wird in der erhaltenen Artikelliste überprüft, ob alle Artikel reserviert wurden. Wenn das nicht zutrifft, wird die Bestellung in der Datenbank mit der aktualisierten Liste aktualisiert. Wenn alle Artikel reserviert wurden, wird der Prozess «Rechnung erstellen» ausgelöst.

Rechnung erstellen: Dieser Prozess wird erst ausgelöst, wenn alle Artikel einer Bestellung verfügbar sind und für diese reserviert wurden. Die Rechnung wird dabei aus den Informationen der Bestellbestätigung generiert, da diese den Kunden als auch die bestellten Artikel inkl. deren Preise beinhalten, und dann an das Kundenmanagement asynchron zugestellt bei dem dann auch deren Persistierung erfolgt.

Datenabfragen: Dabei werden einfache Read-Operationen auf der Datenbank des Bestellservices gemacht. Dazu gehören das Suchen von Bestellungen und Bestellbestätigungen nach deren ID. Bei gefundenen Einheiten werden diese dem Aufrufer zurückgesendet. Da solche Anfragen durch das UI kommen, verlaufen diese immer synchron.

7.2.3. Daten

Beim Bestellmanagement wird alles gehandhabt, was die Bestellung angeht. Darauf werden folgende Einheiten verwaltet:

Bestellung	- ID - Datum - Status - KundenID - Artikelliste
Bestellbestätigung	- ID - Datum - BestellungsID - KundenID - Kundenname - Artikelliste - Preis

Tabelle 6: Einheiten des Bestellmanagements

Bestellung: Bei der Bestellung wird die Verknüpfung zwischen den Artikeln und dem Kunden gemacht. Mit dem Status-Attribut wir festgelegt, in welchem Status sich die Bestellung befindet (Ordered, Delivered, Waiting for product, Cancelled). In der Artikelliste sind alle bestellten Artikel vorhanden welche jeweils als Attribute die ID, den Preis und die Verfügbarkeit beinhalten.

Bestellbestätigung: Diese Einheit ist vor allem für das Userinterface relevant. Diese wird erstellt, wenn eine Bestellung erfolgreich getätigt wurde. Sie wird lediglich einmal erstellt und kann auch nicht aktualisiert werden. Diese wird so persistiert, damit bei einer Abfrage der Bestellbestätigung die Daten bei den anderen Services nicht nochmals angefragt werden müssen.

Es ist wichtig zu erwähnen, dass lediglich die wichtigsten Kundenattribute hier gespeichert werden, da die Kunden im Kundenmanagement verwaltet werden. Beim Austausch von Daten werden meist lediglich die entsprechenden IDs mitgegeben.

7.2.4. Aufbau

Anhand des folgenden vereinfachten Klassendiagramms soll aufgezeigt werden, wie das Bestellmanagement aufgebaut ist:

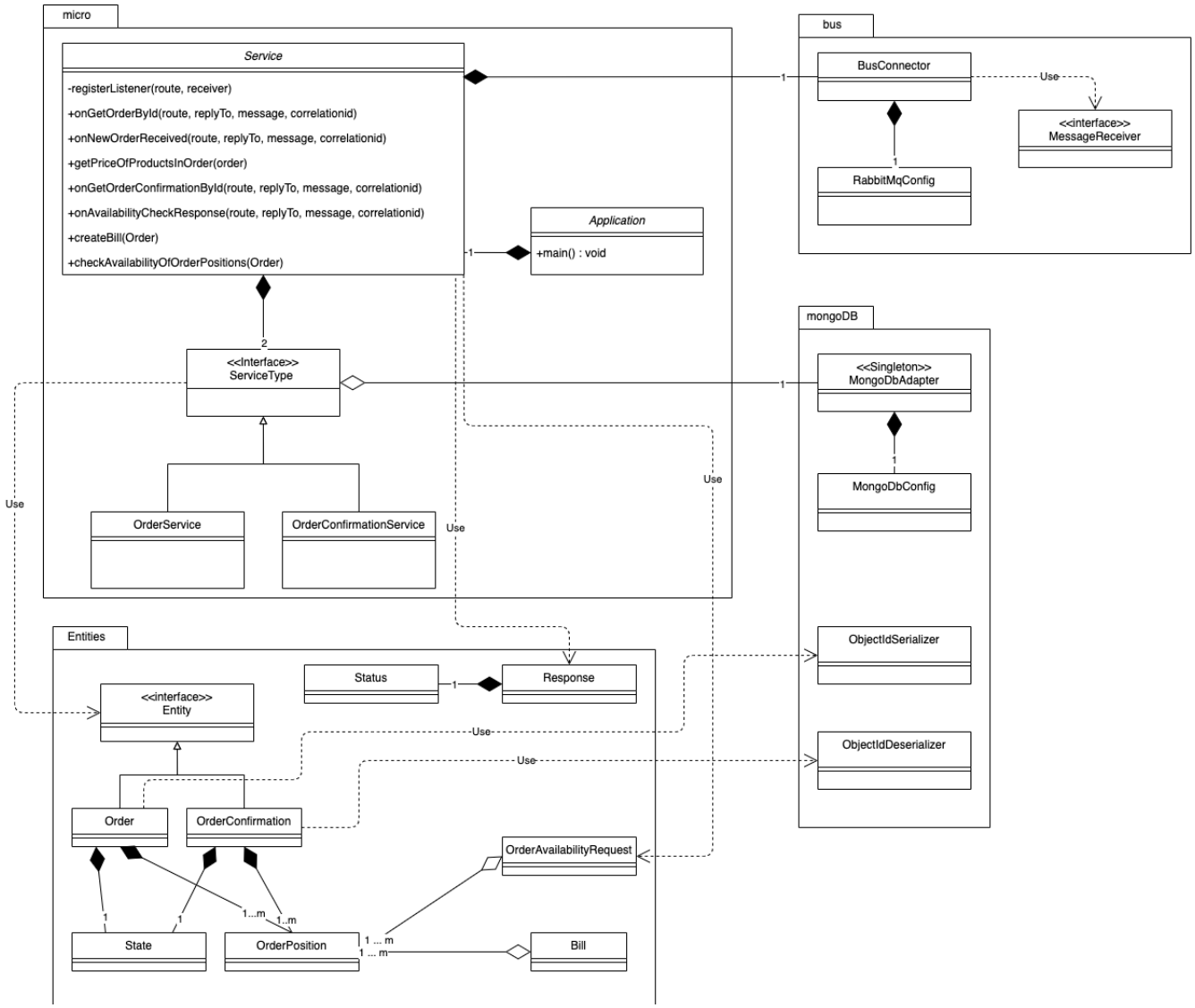


Abbildung 8: Klassendiagramm des Bestellservices

7.2.4.1. Package «micro»

Application: Diese Klasse beinhaltet die main-Methode bei der ein Service-Object instanziiert wird.

Service: Dieser beinhaltet den Hauptstrang des Bestellservices. Im Konstruktor wird der BusConnector initialisiert, die einzelnen Queues erstellt und die entsprechenden Listener

mittels des MessageReceiver aufgestartet. Dabei werden vier Queues gestartet, welche in der späteren Queueübersicht ersichtlich sind.

ServiceType (OrderService und OrderConfirmationService): Während der Service die Messages empfängt, wird die Verarbeitung der Entities an den entsprechenden ServiceType (OrderService oder OrderConfirmationService) weiter delegiert welche jeweils über den MongoDbAdapter mit der Datenbank interagieren. Diese beiden Services arbeiten jeweils auf einer eigenen Collection.

7.2.4.2. Package «bus»

BusConnector: An dieser Klasse wurde nichts geändert. Man hat sich dabei an der gegebenen Implementation bedient.

RabbitMqConfig: An dieser Klasse wurde nichts geändert. Man hat sich dabei an der gegebenen Implementation bedient.

MessageReceiver: Hier wurde die Methode «onMessageReceived» mit dem formalen Parameter «correlationID» ergänzt.

7.2.4.3. Package «mongoDB»

MongoDbAdapter: Hierbei handelt es sich um ein Singleton welches mit der MongoDB interagiert. Hierauf wurde der MongoClient aufgebaut. Darauf werden alle Datenbankoperationen ausgeführt. Da dieser Adapter für jeden ServiceType kompatibel sein muss, arbeitet dieser mit BSON-Dokumenten. Auf dem ServiceType-Interface wurden deshalb default-Methoden mit entsprechenden Convertern implementiert.

MongoDbConfig: Dieses Objekt wird durch den MongoDbAdapter instanziiert um die MongoDB-Configurationen zu laden und eine Verbindung mit der Datenbank herzustellen.

ObjectIdSerializer: Hiermit werden die Entities, welche ein ObjectId-Objekt beinhalten, mit einem eigenen Serialisierer ausgestattet.

ObjectIdDeserializer: Hiermit werden die Entitites, welche ein ObjectId-Objekt beinhalten, mit einem eigenen Deserialisierer ausgestattet.

7.2.4.4. Package «entities»

Entity: Interface, welches alle Objekte, welche in der Datenbank persistiert werden sollen, implementieren müssen.

Order: Objekt, mit dem alle Daten einer Bestellung verwaltet werden.

OrderConfirmation: Objekt, mit dem die Bestellbestätigung für den Kunden repräsentiert wird.

State: Dabei handelt es sich um ein Enum, mit dem der Status der Bestellung beschrieben wird.

OrderPosition: Dabei handelt es sich um einen bestellten Artikel bei dem die ID des Artikels, die geforderte Anzahl und die aktuelle Verfügbarkeit des Artikels repräsentiert wird.

OrderAvailabilityRequest: Mit diesem Objekt werden Requests an das Filiallager zugestellt um die bestellten Artikel zu überprüfen.

Bill: Damit wird die Rechnung für den Kunden erstellt, welche dann an das Kundenmanagement zugestellt wird.

Response: Dabei handelt es sich um ein Objekt, mit dem Antworten an den Gateway zugestellt werden. Diese beinhaltet einen Status, eine Message und Daten.

Status: Dabei handelt es sich um ein Enum mit dem der Status der Verarbeitung beschrieben wird und im Response-Objekt gesetzt wird.

7.3. Filiallager

7.3.1. Einführung

Gemäss Aufgabenstellung hat jede Filiale ein eigenes Filiallager mit Produkten. Es wird bei einer Order genutzt und kann automatisch Nachbestellungen asynchron ausführen. Zusätzlich können Artikel via Rest dem Sortiment hinzugefügt werden.

7.3.2. Beteiligte Prozesse

Das Filiallager ist an folgenden Prozessen beteiligt:

- Bestellung ausführen
- Artikelverfügbarkeit prüfen
- Automatische Nachbestellung
- Artikel dem Bestand hinzufügen

Bestellung ausführen

Der Prozess «Bestellung ausführen» wurde weiter oben bereits dokumentiert. Das Filiallager stellt dabei die Abfrage der Anzahl verfügbaren Artikel und Preis zur Verfügung.

Artikelverfügbarkeit prüfen

Das Filiallager sendet entweder ein True zurück, falls der Artikel in genügender Anzahl vorhanden ist oder ein False. In diesem Fall wird direkt eine Nachbestellung ausgeführt. Die Antwort wird im Ordermanagement weiterverarbeitet.

Automatische Nachbestellung

Das Filiallager überprüft bei jeder Bestellung, ob eine Mindestanzahl vorhanden ist. Falls dies nicht der Fall ist, wird eine neue Nachbestellung asynchron beim Reorder Service beantragt.

Artikel dem Bestand hinzufügen

Im Reorder Service werden Nachbestellungen getätigt. Sobald diese eingegangen sind und alle Artikel nachbestellt, kann die Nachbestellung dem Filiallager hinzugefügt werden. Dies wird via Rest von Datentypist durchgeführt. Anschliessend wird die Anzahl nachbestellter Artikel zu der bestehenden Anzahl im Filiallager addiert.

7.3.3. Daten

Artikel	• ID • Name • Preis • Anzahl im lokalen Lager • Kategorie
Kategorie	• ID • Name

Tabelle 7: Einheiten des Filiallagers

7.3.4. Aufbau

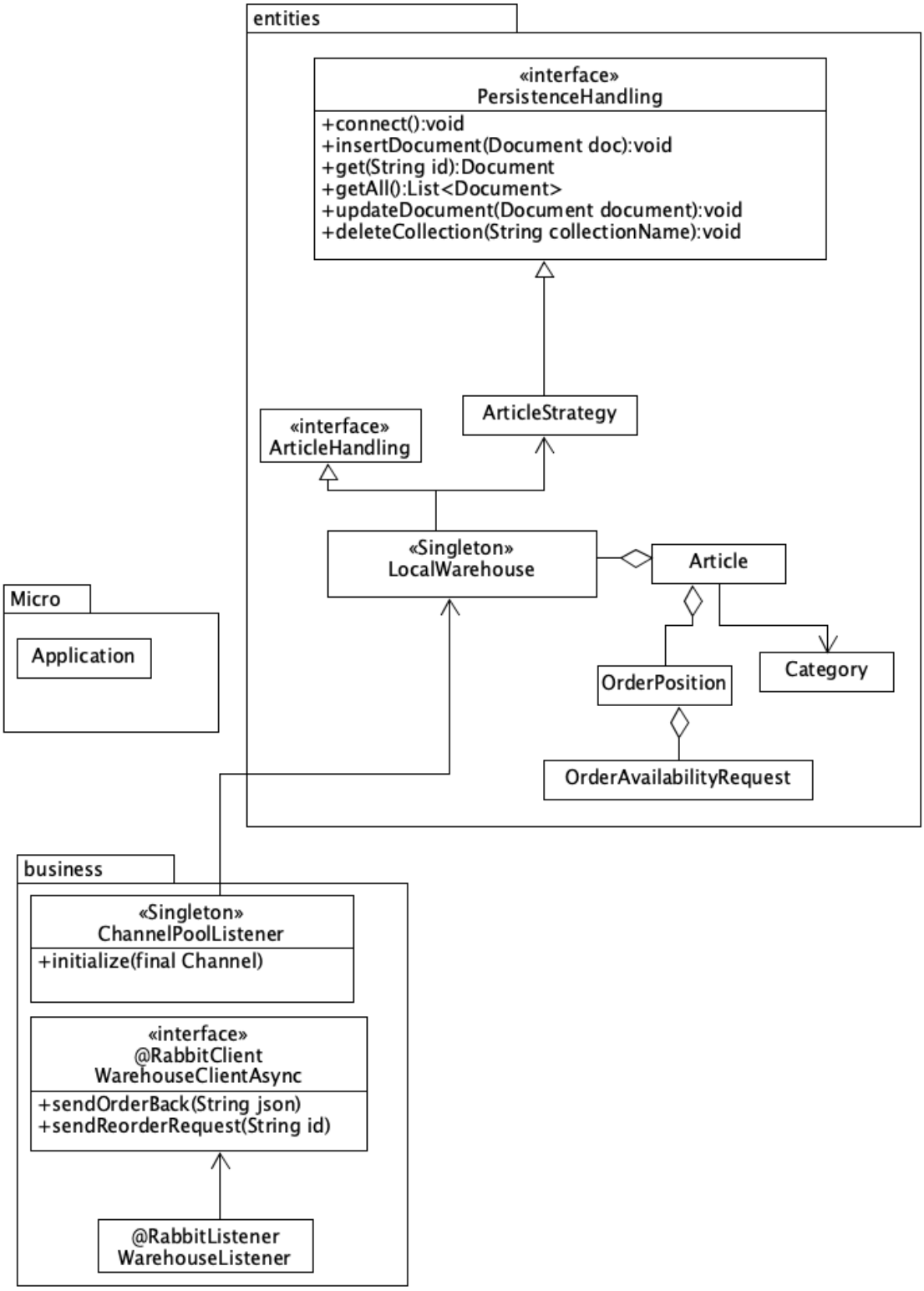


Abbildung 9: Klassendiagramm zum Filiallager

7.3.4.1. Beschreibung

Das obige Klassendiagramm zeigt den Aufbau des lokalen Lagers. Der Microservice wurde vollkommen mit Micronaut aufgebaut.

Package Business

Im Package business wird die Kommunikation mit RabbitMQ gemacht. Der WarehouseListener fungiert dabei als Listener auf den jeweiligen Queues des Lagers. Sofern eine Message in die entsprechende Queue kommt, wird die jeweilige Methode ausgeführt. Zusätzlich wird in dieser Klasse auch eine Antwort zurück auf die Queue geschrieben, falls es sich um eine synchrone Kommunikation handelt. Dies wird vorallem bei Anfragen von der Rest-Schnittstelle gebraucht, jedoch auch zur Kommunikation mit den anderen Services. Durch die Annotation @Singleton wird der Listener von Micronaut als Singleton implementiert.

In der Klasse ChannelPoolListener werden die jeweiligen Queues des Microservices erstellt.

Die Klasse/Interface WarehouseListener, bzw. WarehouseClientAsync wurde im Klassendiagramm umbenannt. In der Software heissen diese ProductClientAsync, bzw. ProductListener.

Package Entities

Im Package Entities werden die Anfragen vom ChannelPoolListener gehandhabt. Neue Artikel können via Rest hinzugefügt, gelöscht oder verändert werden.

Zusätzlich werde hier Orders erstellt und deren Verfügbarkeit überprüft, ehe sie wieder zurückgesendet werden.

Durch den Einsatz des Strategy Pattern ist eine Erweiterung sehr einfach möglich. Es würde es uns ermöglichen, verschiedene Typen in anderen Datenbanken, sogar Datenbanktypen zu persistieren. Denkbar wäre eine Dokumentdatenbank in der einten Strategy und eine Relationale Datenbank in einer anderen.

Package Application

Die Klasse Application dient als Startpunkt des Microservices. Mittels Micronaut wird dieser gestartet und startet anschliessend mittels «Beans» das Listening auf die RabbitMQ Queues.

7.4. Reorder

7.4.1. Einführung

Der Reorder Service kümmert sich um die Kommunikation mit dem zentralen Warehouse. Es ist via zentrales Lager oder auch von der Rest-Schnittstelle ansprechbar und kommuniziert ausschliesslich asynchron mit dem lokalen Lager.

7.4.2. Beteiligte Prozesse

Von diesem Service werden folgende Prozesse bearbeitet:

- Automatische Nachbestellung
- Manuelle Nachbestellung
- Artikel dem Lager hinzufügen(Typist)

Automatische Nachbestellung

Wird bei einer Nachbestellung von lokalen Lager bemerkt, dass eine Nachbestellung nötig ist, so wird diese automatisch ausgelöst. Es wird eine asynchrone Nachricht an den Reorder Service gesendet. Dieses bestellt anschliessend die Artikel im Lager nach. Sobald die Artikel eingetroffen sind, wird der Status-Wert auf True gesetzt und die Bestellung kann von Typist hinzugefügt werden.

Manuelle Nachbestellung

Die manuelle Bestellung erfolgt von der Rest-Schnittstelle aus. Sie wird mittels Post erstellt. Der weitere Vorgang bleibt identisch zu oben.

Artikel dem Lager hinzufügen

Sobald alle Artikel dem Lager hinzugefügt wurden, kann die Bestellung vom Typist dem Lager hinzugefügt werden. Die nachbestellte Anzahl Artikel wird zum Bestand addiert.

7.4.3. Aufbau

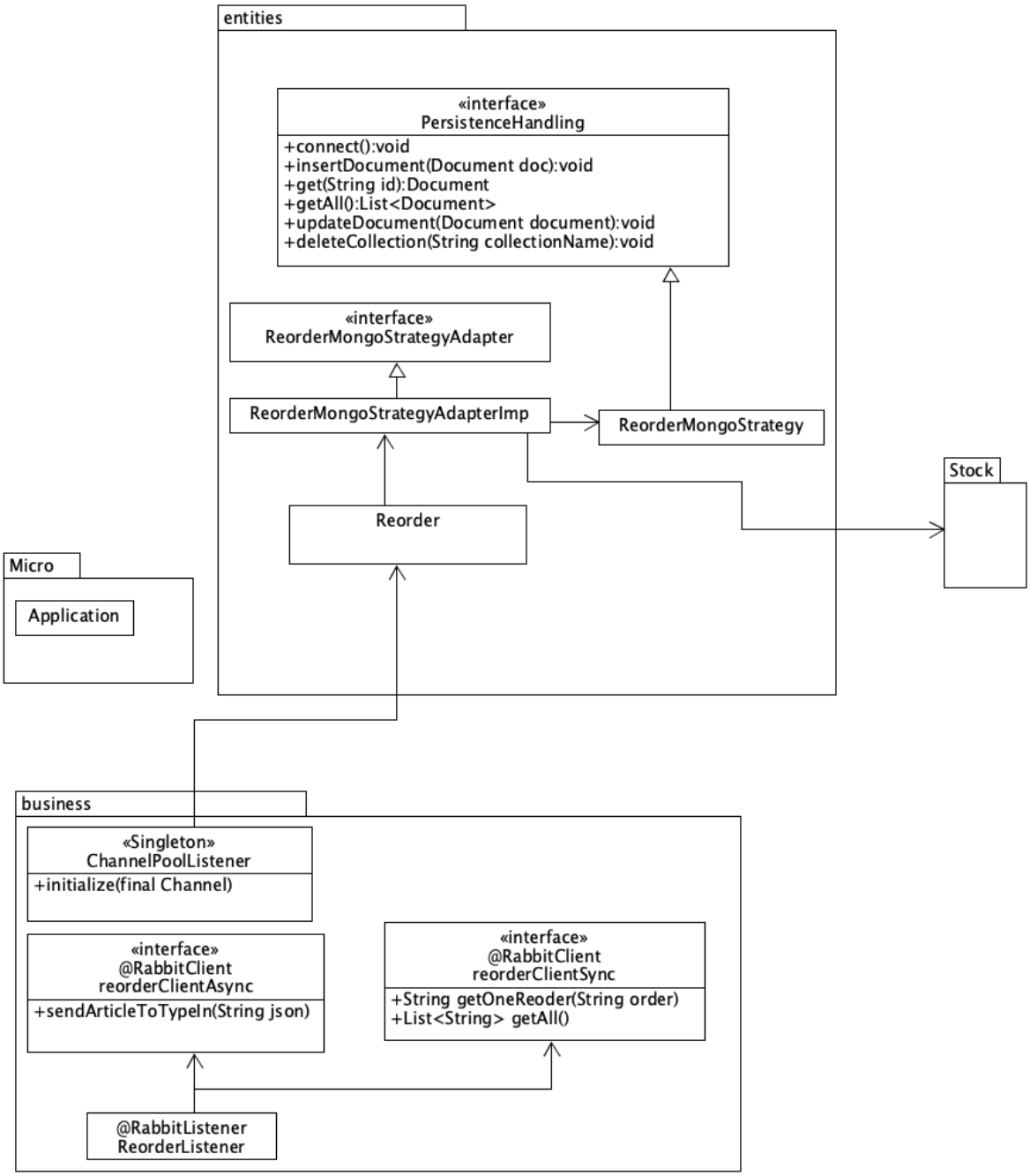


Abbildung 10: Klassendiagramm zum Reorder

Package Business

Das Package Business unterscheidet sich nicht von dem Package vom lokalen Lager. Einzig das es hier zwei Clients gibt, einer für asynchrone und eine für asynchrone Aufrufe.

Package Micro

Dieses Package ist identisch zu dem Lokalen Lager Service. Es ist weiter oben bereits beschrieben.

Package Entities

Wie auch beim lokalen Lager kommt hier das Strategy Pattern zum Einsatz. Die momentan genutzte Implementierung ist dabei für eine Dokumentdatenbank ausgelegt. Zusätzlich wurde hier noch ein Adapter verwendet. MongoDb arbeitet mit bson Dokumenten, der Adapter wandelt String von Messages in solche um, bzw. macht aus bson-Dokumenten wieder Strings.

7.5. Gateway

7.5.1. Einführung

In dieser Applikation soll der Gateway als Schnittstelle für alle Clientanfragen fungieren. Dabei soll dieser alle Anfragen über das RabbitMQ an die entsprechenden Services weiterleiten. Gemäss dem http-Standard, da http-Anfragen immer synchron ablaufen, wurden die RabbitMQ-Messages vom Gateway aus synchron implementiert. Der Gateway wurde anhand des Templates, welches mit «Micronaut» implementiert wurde, aufgebaut. Für die Anbindung des Gateways mit dem Messaging-System wurde das RabbitMQ-Paket von «Micronaut» verwendet.

7.5.2. Aufbau

Der Gateway beinhaltet mehrere Controller welche in der untenstehenden Liste ersichtlich sind. Auf die Handhabung der http-Requests wird in der Schnittstellenspezifikation genauer eingegangen.

Name des Controllers	**Route des Controllers**	**http-Requests**
OrderController	/api/v1/order	POST: / GET: /{?orderID}
CustomerController	/api/v1/customers	POST: / GET: /{?id} GET: /{?last_name}
BillController	/api/v1/customers/bills	GET: /{?billID}
OrderConfirmationController	/api/v1/order/confirmation	GET: /{?orderConfirmationID}
ReorderController	/api/v1/ typeIn	POST: / PUT: /
StorageController	/api/v1/storage	POST: / PUT: /{?storageItemID} GET: /{?storageItemID}

Tabelle 8: Übersicht der Controller

Des Weiteren wurde die Anbindung an das RabbitMQ mit dem Micronaut-RabbitMQ-Packet sichergestellt. Hierfür wurde überall als Exchange der «g03» deklariert. In der Spalte «Binding» sind die Routing-Keys deklariert anhand dieser die gewünschten Queues adressiert werden.

RabbitMQ-Client	**Methoden / Binding**	**Binding**
CustomerClientSync	getAll()	customer.getall
	getOneByID()	customer.getOneById
	findByLastName()	customer.findByLastName
	upsert()	customer.upsert
	getOneBill()	customer.getOneBill
OrderClientSync	getOrderByID()	order.getone
	create()	order.neworder
	getOrderConfirmationID()	order.getoneconfirmation
StorageClientSync	getOneArticle()	storage.one
	getAllArticles()	storage.all
	insertArticle()	storage.insert
	updateArticle()	storage.updateWhole
	updateArticleNumber()	storage.updateNumber
ReorderClientSync	newReorder()	reorder.insert
	getOneReorder()	reorder.getOne
	getAllReorder()	reorder.getAll
	setAsInserted()	reorder.update

Tabelle 9: Übersicht der RabbitMQ-Clients

Bei diesen RabbitMQ-Clients handelt es sich lediglich um Interfaces, welche keine konkrete Implementation beinhalten. Diese werden den Controllern über den Konstruktor übergeben wodurch die Messages dann automatisch dem Exchange mit dem entsprechenden Routing-Key übergeben werden.

8. Übersicht der Queues

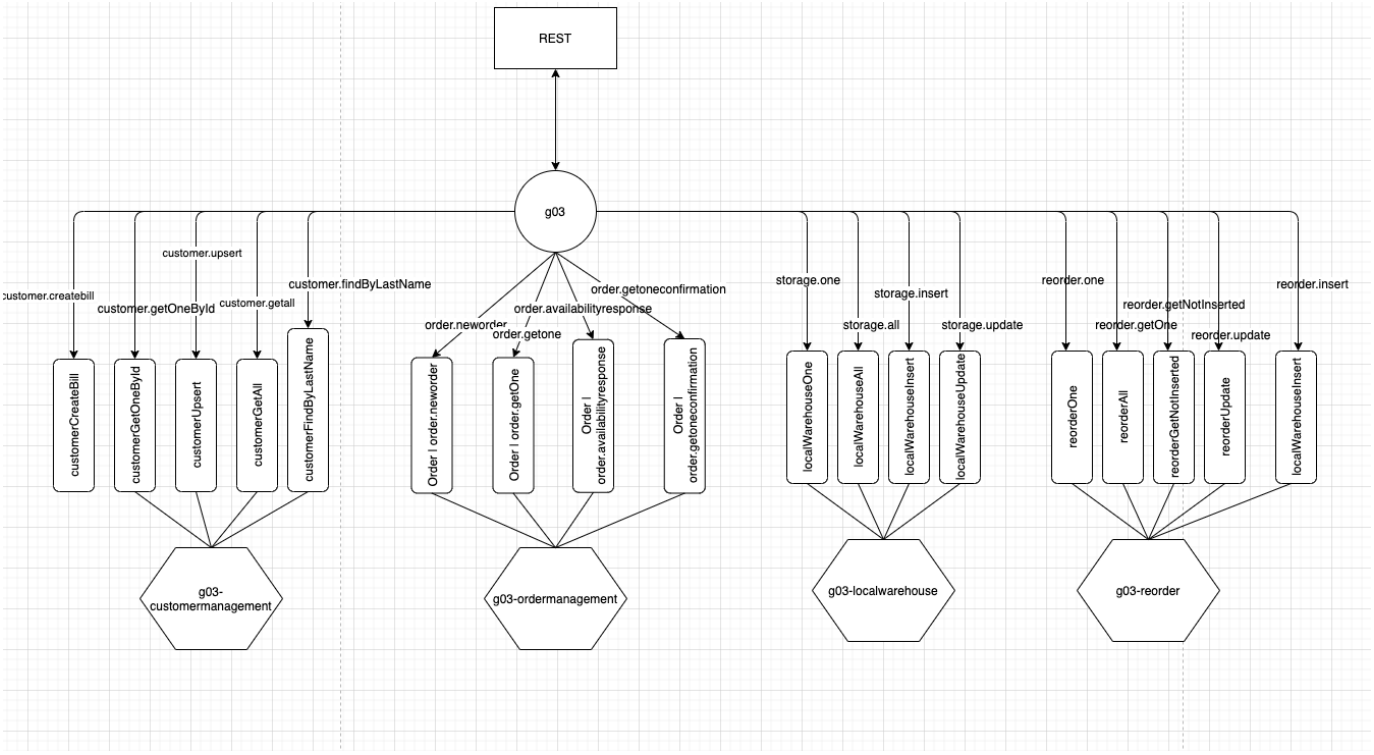


Abbildung 11: Übersicht über die Queues auf dem Exchange g03

9. Schnittstellenspezifikation

9.1. Einführung

In diesem Kapitel werden die verschiedenen Schnittstellen erläutert wobei auf jeden einzelnen Service als auch den Gateway eingegangen wird. Dabei wird zwischen Class-based APIs, der REST-API und der Schnittstellen für das Messaging-System mit RabbitMQ unterschieden.

Grundsätzlich wird der ganze Datenaustausch zwischen den Microservices als auch der REST mittels JSON-Dokumenten gemacht. Man ist somit nicht zwingend an die Programmiersprache «Java» gebunden. Des Weiteren erfolgt der Nachrichtenaustausch über das Messagingsystem «RabbitMQ». Es wird in der Schnittstellenspezifikation definiert, wie eine Message in einer bestimmten Queue aufgebaut sein muss, damit diese erfolgreich eingelesen und auf dem Service verarbeitet wird.

9.2. Gateway

Diese REST-API wurde so aufgebaut, dass diese die erhaltenen Messages lediglich weiterleitet. Dies wurde mit Absicht gemacht, um nicht zu viel Logik darauf aufzubauen und den Fokus mehr auf die einzelnen Services zu setzen. Deshalb müssen bereits auf dem Client die Messages korrekt, gemäss den Schnittstellenspezifikationen der einzelnen Services, an die REST versendet werden. Da die REST keine Überprüfung der übergebenen Parameter macht, kann grundsätzlich alles an diese versendet werden, solange man die korrekte URI mit den entsprechenden http-Requests verwendet. Auf den Aufbau der Messages wird aus Gründen der Doppelspurigkeit nicht hier, sondern in den Schnittstellenspezifikationen der einzelnen Services eingegangen. Hier soll lediglich aufzeigt werden welche Prozesse über welchen Controller angestossen werden.

9.2.1. Kunden

Titel	Neuen Kunden erfassen
Zweck / Semantik	Hierbei soll ein neuer Kunde erfasst werden. Die Message muss an das Kundenmanagement zugestellt werden. Der Message-Aufbau und die Anforderung wird in der Spezifikation der Queue «Customers | customer.upsert» beschrieben. Als Antwort erhält man einen http-Status-Code mit einer Meldung ob die Erfassung erfolgreich war und wenn ja, ist noch eine Kunde mitenthalten.
URL	restinterface.appe-g03.el.eee.intern/api/v1/customers
http-Request	POST
Voraussetzung	-
Rückgabe	HTTP-Status-Code inkl. Nachricht und Daten
Protokoll	Synchron

Tabelle 10: REST-Spezifikation für "Kunde erfassen"

Titel	Kunde über ID suchen
Zweck / Semantik	Hierbei soll ein Kunde anhand der übergebenen ID gesucht und zurückgegeben werden. Die Message muss an das Kundenmanagement zugestellt werden. Der Message-Aufbau und die Anforderung wird in der Spezifikation der Queue «Customer | customer.getOneById» beschrieben. Als Antwort erhält man einen http-Status-Code mit einer Meldung, ob die Suche erfolgreich war und wenn ja, ist der gesuchte Kunde mitenthalten.
URL	restinterface.appe-g03.el.eee.intern/api/v1/customers/<customerId>
http-Request	GET
Voraussetzung	- Parameter «customerId» muss in der URL der GET-Anfrage mitenthalten sein.
Rückgabe	HTTP-Status-Code inkl. Nachricht und Daten
Protokoll	Synchron

Tabelle 11: REST-Spezifikation für "Kunde via ID suchen"

Titel	Kunde über Nachname suchen
Zweck / Semantik	Hierbei soll ein Kunde anhand des Nachnamens gesucht und zurückgegeben werden. Die Message muss an das Kundenmanagement zugestellt werden. Der Message-Aufbau und die Anforderung wird in der Spezifikation der Queue «Customer | customer.findByLastName» beschrieben. Als Antwort erhält man einen http-Status-Code mit einer Meldung, ob die Suche erfolgreich war und wenn ja, ist eine Liste aller Kunden mit diesem Nachnamen mitenthalten.
URL	restinterface.appe-g03.el.eee.intern/api/v1/customers/ ?last_name=<Nachname>
http-Request	GET
Voraussetzung	- Parameter «Nachname» muss in der URL der GET-Anfrage mitenthalten sein.
Rückgabe	HTTP-Status-Code inkl. Nachricht und Daten
Protokoll	Synchron

Tabelle 12: REST-Spezifikation für "Kunde via ID suchen"

Titel	Alle Kunden zurückgeben
Zweck / Semantik	Hierbei sollen alle Kunden, die im Kundenservice persistiert sind, zurückgegeben werden. Die Message muss an das Kundenmanagement zugestellt werden. Der Message-Aufbau und die Anforderung wird in der Spezifikation der Queue «Customer | customer.getAll» beschrieben. Als Antwort erhält man einen http-Status-Code mit einer Meldung, ob die Suche erfolgreich war und wenn ja, ist eine Liste aller Kunden mit diesem Nachnamen mitenthalten.
URL	restinterface.appe-g03.el.eee.intern/api/v1/customers/
http-Request	GET
Voraussetzung	
Rückgabe	HTTP-Status-Code inkl. Nachricht und Daten
Protokoll	Synchron

Tabelle 13: REST-Spezifikation für "Kunde via ID suchen"

9.2.2. Rechnungen

Titel	Rechnung über Rechnungsnummer suchen
Zweck / Semantik	Hierbei soll ein Kunde anhand des Nachnamens gesucht und zurückgegeben werden. Die Message muss an das Kundenmanagement zugestellt werden. Der Message-Aufbau und die Anforderung wird in der Spezifikation der Queue «Customer | customer.findByLastName» beschrieben. Als Antwort erhält man einen http-Status-Code mit einer Meldung, ob die Suche erfolgreich war und wenn ja, ist eine Liste aller Kunden mit diesem Nachnamen mitenthalten.
URL	http://restinterface.appe-g03.el.eee.intern/api/v1/customers/bills/?billID=<Rechnungsnummer>
http-Request	GET
Voraussetzung	- Parameter «Rechnungsnummer» muss in der URL der GET-Anfrage mitenthalten sein.
Rückgabe	HTTP-Status-Code inkl. Nachricht und Daten
Protokoll	Synchron

Tabelle 14: REST-Spezifikation für " Rechnung über Rechnungsnummer suchen"

9.2.3. Bestellungen

Titel	Neue Bestellung erfassen
Zweck / Semantik	Hierbei soll eine neue Bestellung erfasst werden. Die Message muss an das Bestellmanagement zugestellt werden. Der Message-Aufbau und die Anforderung wird in der Spezifikation der Queue «Order | order.neworder» beschrieben. Als Antwort erhält man einen http-Status-Code mit einer Meldung ob die Erfassung erfolgreich war und wenn ja, ist noch eine Bestellbestätigung mitenthalten.
URL	restinterface.appe-g03.el.eee.intern/api/v1/order
http-Request	POST

Voraussetzung	-Kunden im Body als JSON
	Beispiel:
	`{`
	`"first_name": "Skyler",`
	`"last_name": "White",`
	`"street": "Wyoming Bldv",`
	`"number": "1501",`
	`"zip": "8001",`
	`"city": "Albuquerque",`
	`"email": "walter.white@breakingbad.com",`

	<pre>"phone": "+41787894502"
	}</pre>

Rückgabe	HTTP-Status-Code inkl. Nachricht und Daten
Protokoll	Synchron

Tabelle 15: REST-Spezifikation für "Bestellung erfassen"

Titel	Bestellung über ID suchen
Zweck / Semantik	Hierbei soll eine Bestellung anhand der übergebenen ID gesucht und zurückgegeben werden. Die Message muss an das Bestellmanagement zugestellt werden. Der Message-Aufbau und die Anforderung wird in der Spezifikation der Queue «Order | order.getone» beschrieben. Als Antwort erhält man einen http-Status-Code mit einer Meldung ob die Erfassung erfolgreich war und wenn ja, ist noch eine Bestellung mitenthalten.
URL	restinterface.appe-g03.el.eee.intern/api/v1/order?orderID=
http-Request	GET
Voraussetzung	- Parameter «orderID» muss in der GET-Anfrage mitenthalten sein.
Rückgabe	HTTP-Status-Code inkl. Nachricht und Daten
Protokoll	Synchron

Tabelle 16: REST-Spezifikation für "Bestellung suchen"

Titel	Bestellbestätigung über ID suchen
Zweck / Semantik	Hierbei soll eine Bestellbestätigung anhand der übergebenen ID gesucht und zurückgegeben werden. Die Message muss an das Bestellmanagement zugestellt werden. Der Message-Aufbau und die Anforderung wird in der Spezifikation der Queue «Order | order.getoneconfirmation» beschrieben. Als Antwort erhält man einen http-Status-Code mit einer Meldung, ob die Bestellbestätigung gefunden wurde und wenn ja, ist diese noch mitenthalten.
URL	restinterface.appe-g03.el.eee.intern/api/v1/order/confirmation?orderConfirmationID=
http-Request	GET
Voraussetzung	- Parameter «oderConfirmationID» muss in der GET-Anfrage mitenthalten sein.
Rückgabe	HTTP-Status-Code inkl. Nachricht und Daten
Protokoll	Synchron

Tabelle 17: REST-Spezifikation für "Bestellbestätigung suchen"

9.2.4. Filiallager

Titel	Artikel suchen
Zweck / Semantik	Hierbei soll ein Artikel anhand der übergebenen ID gesucht und zurückgegeben werden. Die Message muss an das Filiallager zugestellt werden. Als Antwort erhält man einen http-Status-Code mit einer Meldung, ob der Artikel gefunden wurde und wenn ja, ist diese noch mitenthalten. Die storageItemID kann dabei leer gelassen werden, dann erhält man alle Artikel im Lager oder mit einer ID gefüllt werden, sodass man einen bestimmten Artikel erhält.
URL	restinterface.appe-g03.el.eee.intern/api/v1/storage?storageItemID=
http-Request	GET
Voraussetzung	- Parameter «storageItemID» kann leer sein
Rückgabe	HTTP-Status-Code inkl. Nachricht und Daten Beispiel: ["{ \"_id\" : \"765f42cc-c319-41c0-9af6-f7c78cfdff7d\", \"name\" : \"demoArticle7056\", \"numberInStock\" : 191, \"price\" : 235.0, \"category\" : { \"ID\" : 2, \"name\" : \"Keyboards\" } }",
Protokoll	Synchron

Tabelle 18: REST-Spezifikation für "Artikel suchen"

Titel	Artikel einfügen
Zweck / Semantik	Ein Artikel soll dem Lager hinzugefügt werden. Dazu muss der Artikel als JSON im Body übergeben werden. Als Antwort erhält man einen http Code sowie eine Bestätigung, falls das Erfassen erfolgreich war. Der genaue Aufbau der Message wird bei der queue localWarehouseInsert beschrieben
URL	restinterface.appe-g03.el.eee.intern/api/v1/storage?storageItemID=
http-Request	POST
Voraussetzung	- Artikel im Body als JSON Beispiel: { "_id": "765faaac-c319-41c0-9af6-f7c78cfdff7d",

	"name": "spray",
	"numberInStock": 31,
	"price": 0,
	"category": {
	"ID": 2,
	"name": "Keyboards"
	}
	}

Rückgabe	HTTP-Status-Code inkl. Antwort
Protokoll	Synchron

Tabelle 19: REST-Spezifikation für "Artikel hinzufügen""

9.2.5. Reorder

Titel	Nachbestellung suchen
Zweck / Semantik	Eine Nachbestellung kann auf 3 verschiedene Arten gesucht werden: Ist der reorderID leer, so werden alle reorders zurückgegeben, ist die ID gesetzt wird die Betreffende zurückgegeben. Es besteht auch noch die Möglichkeit, den Wert der ID auf False zu setzen und somit alle reorders zu bekommen, welche dem Filiallager noch nicht hinzugefügt wurden
URL	restinterface.appe-g03.el.eee.intern/api/v1/typeIn?reorderId=
http-Request	GET
Voraussetzung	
Rückgabe	HTTP-Status-Code inkl. Nachricht und Daten Beispiel: "{ \"_id\" : \"sdsdsdxv-sdsdsdxccd-xx-fgdfdfsd-jsshdjs\", \"articles\" : { \"765f42cc-c319-41c0-9af6-f7c78cfdff7d\" : 100 } }"
Protokoll	Synchron

Tabelle 20: REST-Spezifikation für "Nachbestellung suchen"

Titel	Nachbestellung erstellen

Zweck / Semantik	Es soll eine neue Nachbestellung erstellt werden. Dazu muss mittels Post die Reorder dem Service mitgeteilt werden. Die genaue Spezifikation dieser Message ist in der Beschreibung der Queue unter zu finden
URL	restinterface.appe-g03.el.eee.intern/api/v1/typeIn
http-Request	POST
Voraussetzung	Nachbestellung im Body als JSON `"{ \"_id\" : \"sdsdsdxv-sdsdsdxccd-xx-fgdfdfsd-jsshdjs\", \"articles\" : { \"765f42cc-c319-41c0-9af6-f7c78cfdff7d\" : 100 } }"`
Rückgabe	HTTP-Status-Code inkl. Antwort
Protokoll	Synchron

Tabelle 21: REST-Spezifikation für "Nachbestellung erstellen"

Titel	Nachbestellung dem Lager hinzufügen
Zweck / Semantik	Eine Nachbestellung soll von Typist dem lokalen Lager hinzugefügt werden. Hierzu wird ein PUT auf die jeweilige Reorder gemacht.
URL	restinterface.appe-g03.el.eee.intern/api/v1/typeIn/reorderID=
http-Request	PUT
Voraussetzung	- reorderID im body Beispiel: 5ec7ff341c46860007da7c19
Rückgabe	HTTP-Status-Code inkl. Antwort
Protokoll	Synchron

Tabelle : REST-Spezifikation für "Nachbestellung dem lokalen Lager hinzufügen "

9.3. Kundenmanagement

9.3.1. Queues

<table>
<tr><th>Queue-Name</th><td>Customer | customer.upsert</td></tr>
<tr><th>Zweck / Semantik</th><td>Auf der Applikation soll ein neuer Kunde erfasst werden und damit für die Bestellung zur Verfügung stehen.

Alternativ kann die Information zu einem bestehenden Kunde über diese Queue aktualisiert werden</td></tr>
<tr><th>Voraussetzung</th><td>- Alle Felder ausser der ID und der Telefonnummer müssen befüllt sein.</td></tr>
<tr><th>Message Aufbau</th><td>- Informationen zum Kunden im JSON-Format, wie im Beispiel aufgelistet.
- Wenn die UUID mitgegeben wird, wird der Kunde aktualisiert anhand der mitgelieferten Daten.
- Wenn keine UUID mitgegeben wird, wird ein neuer Kunde erstellt. Der Kunde wird zusammen mit der neuen ID zurückgegeben.

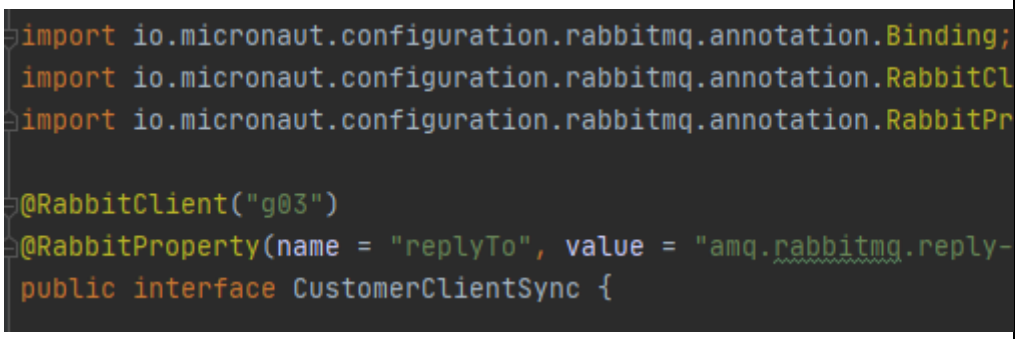
Beispiel Kunde erstellen/updaten:
<pre>{
  "uuid": "eefc125c-6b01-4eb7-a2db-13f93f7891f1"
  "first_name": "Skyler",
  "last_name": "White",
  "street": "Wyoming Bldv",
  "number": "1501",
  "zip": "8001",
  "city": "Albuquerque",
  "email": "walter.white@breakingbad.com",
  "phone": "+41787894502"
}</pre></td></tr>
<tr><th>Rückgabe</th><td>Für die Response verwendet micronaut den Direct-Reply-To (RPC Mechanismus von RabbitMQ.) Dies wird Client Seitig, also auf dem g03-restinterface definiert. Im Interface CustomerClientSynv wird dort der Exchangename, sowie der Replyto Mechniasmus definiert:
<pre>import io.micronaut.configuration.rabbitmq.annotation.Binding;
import io.micronaut.configuration.rabbitmq.annotation.RabbitClient
import io.micronaut.configuration.rabbitmq.annotation.RabbitProper

@RabbitClient("g03")
@RabbitProperty(name = "replyTo", value = "amq.rabbitmq.reply-to")
public interface CustomerClientSync {</pre></td></tr>
</table>

	Für die synchrone Antwort vom Listener wird dies genutzt. Zurückgegeben wird ein CustomerResponse-Objekt welches wie folgt aufgebaut ist: - Success: boolean - Message (String): Information über den Verlauf. Wenn Kunden aktualisiert oder neu erstellt wurden, wird der Kunde im JSON Format zurückgegeben. - Die Antwort **Beispiel Antwort nach Kundeerstellen, updaten:** `{ "customer": { "uuid": "eefc125c-6b01-4eb7-a2db-13f93f7891f1", "first_name": "Jesse", "last_name": "Pinkman", "street": "Street Northwest", "number": "322", "zip": "8003", "city": "Albuquerque", "email": "jesse.pinkman@breakingbad.com", "phone": "+41787894503", "admonitionLevel": "NOTHING" }, "success": true }`  Nicht erfolgreich:  `{ "success": false, "message": "Could not perform upsert. The following fields are missing data: [streetName, streetNumber, city, email, zipCode]" }`
Protokoll	Synchron
Eigenschaften	.

Tabelle 22: Spezifikation für einen neuen Kunden oder für die Aktualisierung eines Kunden

Queue-Name	Customer | customer.findByLastName
Zweck / Semantik	Auf der Applikation soll ein Kunde aufgrund des Nachnamens gefunden werden.
Voraussetzung	- Der Nachname muss in der URL des GET Aufrufes mitgegeben werden.

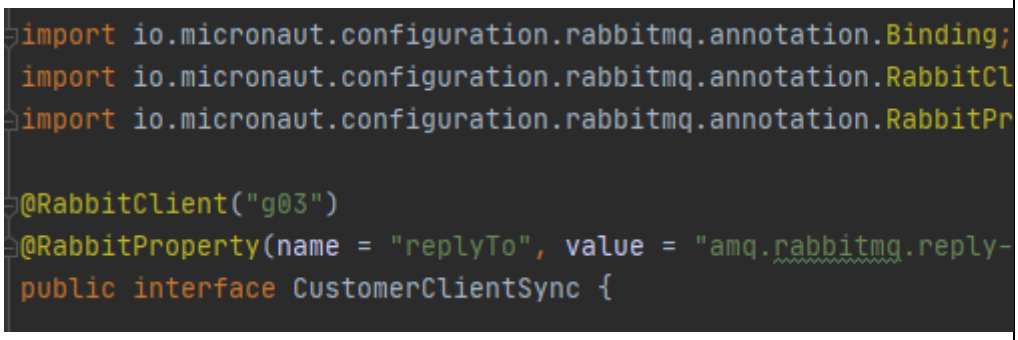
Message Aufbau	`Kein Inhalt`
Rückgabe	Für die Response verwendet micronaut den Direct-Reply-To (RPC Mechanismus von RabbitMQ.) Dies wird Client Seitig, also auf dem g03-restinterface definiert. Im Interface CustomerClientSynv wird dort der Exchangename, sowie der Replyto Mechniasmus definiert:  Für die synchrone Antwort vom Listener wird dies genutzt. Zurückgegeben wird ein CustomerResponse-Objekt welches wie folgt aufgebaut ist: - Success: boolean - Message (String): Information über den Verlauf. Wenn Kunden gefunden werden, wird eine Liste der Kunde mit dem gesuchten Nachnamen im JSON Format zurückgegeben. **Beispiel Antwort nach Kunde per Nachname White suchen:** { "customers": [{ "uuid": "1f15aacc-9e4b-4e73-85db-8943dc2ea18c", "first_name": "Walter", "last_name": "White", "street": "Wyoming Bldv", "number": "1501", "zip": "8002", "city": "Albuquerque", "email": "walter.white@breakingbad.com", "phone": "+41787894502", "admonitionlevel": "FIRST_LEVEL", }, { "uuid": "7e20b173-fd26-43d4-8f61-2e9d452b9605", "first_name": "Skyler", "last_name": "White", "street": "Wyoming Bldv", "number": "1501", "zip": "8001", "city": "Albuquerque", "email": "skyler.white@breakingbad.com", "phone": "+41787894501", "admonitionlevel": "NOTHING",

	<pre> }
	    ],
	    "success": true
	}</pre>

Protokoll	Synchron
Eigenschaften	.

Tabelle 23: Spezifikation für die Suche nach Kunden via Nachname

Queue-Name	Customer | customer.getOneById
Zweck / Semantik	Auf der Applikation soll ein Kunde aufgrund der Kundennummer gefunden werden.
Voraussetzung	- Die Kundennummer muss in der URL des GE Taufrufes mitgegeben werden.
Message Aufbau	`Kein Inhalt`

Rückgabe	Für die Response verwendet micronaut den Direct-Reply-To (RPC Mechanismus von RabbitMQ.) Dies wird Client Seitig, also auf dem g03-restinterface definiert. Im Interface CustomerClientSynv wird dort der Exchangename, sowie der Replyto Mechniasmus definiert:
	 
	Für die synchrone Antwort vom Listener wird dies genutzt.
	Zurückgegeben wird ein CustomerResponse-Objekt welches wie folgt aufgebaut ist:
	- Success: boolean
	- Message (String): Information über den Verlauf. Wenn ein Kunde gefunden werden, wird er im JSON Format zurückgegeben.
	
	Beispiel Antwort nach Kunde per Kundennummer suchen:
	<pre>{
	    "customer": {
	        "uuid": "eefc125c-6b01-4eb7-a2db-13f93f7891f1",</pre>

	"first_name": "Jesse", "last_name": "Pinkman", "street": "Street Northwest", "number": "322", "zip": "8003", "city": "Albuquerque", "email": "jesse.pinkman@breakingbad.com", "phone": "+41787894503", "admonitionLevel": "NOTHING" }, "success": true }
Protokoll	Synchron
Eigenschaften	.

Tabelle 24: Spezifikation für die Suche nach Kunden via Kundennummer

Queue-Name	Customer | customer.getAll
Zweck / Semantik	Auf der Applikation sollen alle Kunden angezeigt werden, die auf dem Kundenservice gespeichert sind.
Voraussetzung	
Message Aufbau	`Kein Inhalt`

Rückgabe	Für die Response verwendet micronaut den Direct-Reply-To (RPC Mechanismus von RabbitMQ.) Dies wird Client Seitig, also auf dem g03-restinterface definiert. Im Interface CustomerClientSynv wird dort der Exchangename, sowie der Reply-to Mechanismus definiert:
	`import io.micronaut.configuration.rabbitmq.annotation.Binding;`
	`import io.micronaut.configuration.rabbitmq.annotation.RabbitClient;`
	`import io.micronaut.configuration.rabbitmq.annotation.RabbitProperty;`
	`@RabbitClient("g03")`
	`@RabbitProperty(name = "replyTo", value = "amq.rabbitmq.reply-to")`
	`public interface CustomerClientSync {`
	Für die synchrone Antwort vom Listener wird dies genutzt.
	Zurückgegeben wird ein CustomerResponse-Objekt welches wie folgt aufgebaut ist:
	- Success: boolean
	- Message (String): Information über den Verlauf. Wenn Kunden gefunden werden, werden sie im JSON Format zurückgegeben.

	Beispiel Antwort nach alle Kunden anfordern:
	{
	"customers": [
	{
	"uuid": "1f15aacc-9e4b-4e73-85db-8943dc2ea18c",
	"first_name": "Walter",
	"last_name": "White",
	"street": "Wyoming Bldv",
	"number": "1501",
	"zip": "8002",
	"city": "Albuquerque",
	"email": "walter.white@breakingbad.com",
	"phone": "+41787894502",
	"admonitionlevel": "FIRST_LEVEL",
	},
	{
	"uuid": "7e20b173-fd26-43d4-8f61-2e9d452b9605",
	"first_name": "Skyler",
	"last_name": "White",
	"street": "Wyoming Bldv",
	"number": "1501",
	"zip": "8001",
	"city": "Albuquerque",
	"email": "skyler.white@breakingbad.com",
	"phone": "+41787894501",
	"admonitionlevel": "NOTHING",
	}
	],
	"success": true
	}

Protokoll	Synchron
Eigenschaften	.

Tabelle 25: Spezifikation für die Suche nach Kunden via getAll

Queue-Name	Customer | customer.createBill
Zweck / Semantik	Zu einer Bestellung soll eine Rechnung erstellt werden. Der Aufruf kommt vom Bestellservice an den Kundenservice.
Voraussetzung	- Kunde muss bereits erfasst sein. - Bestellung muss erfasst sein.
Message Aufbau	Beispiel {"orderID":"5ecbbe074d75d712af925447", "customerID":"7e20b173-fd26-43d4-8f61-2e9d452b9605", "firstName":null, "lastName":null,

	"articleList": [{"articleID":"765f42cc-c319-41c0-9af6-f7c78cfdff7d", "count":3, "available":false,"price":235.0}], "date":null,"fullPrice":705.0,"_id":null}
Rückgabe	
Protokoll	Asynchron
Eigenschaften	.

Tabelle 26: Spezifikation für die Erstellung einer Rechnung

Queue-Name	Customer | customer.getOneBillByID
Zweck / Semantik	Eine Rechnung soll aufgrund der Rechnungsnummer gefunden werden.
Voraussetzung	- Kunde muss bereits erfasst sein. - Bestellung muss erfasst sein. - Rechnung muss vorhanden sein. - ID muss in der URL mitgegeben werden.
Message Aufbau	
Rückgabe	Die Antwort kommt vom CustomerListener. Zurückgegeben wird ein CustomerResponse-Objekt welches wie folgt aufgebaut ist: - Success: boolean - Message (String): Information über den Verlauf. Wenn ein Kunde gefunden werden, wird er im JSON Format zurückgegeben. - Wenn keine Rechnung gefunden wird, wird null zurückgegeben. **Beispiel Rechnung gefunden:** { "customer": null, "customers": null, "success": true, "message": "CustomerListener retrieved bill: {\"orderID\":\"5ecbbe074d75d712af925447\", \"customerID\":\"7e20b173-fd26-43d4-8f61-2e9d452b9605\", \"firstName\":\"Skyler\", \"lastName\":\"White\", \"articleList\":[{\"articleID\":null, \"count\":0, \"available\":false, \"price\":235.0}], \"date\":1590724108131, \"fullPrice\":705.0,

	\"_id\":{\"$oid\":\"5ed0860c656e6445649db9a0\"}}"
	}
	Beispiel keine Rechnung gefunden:
	{
	"customer": null,
	"customers": null,
	"success": true,
	"message": "CustomerListener retrieved bill: null"
	}

Protokoll	Synchron
Eigenschaften	.

Tabelle 27: Spezifikation für die Suche einer Rechnung mittels Rechnungsnummer

9.4. Bestellmanagement

9.4.1. Queues

Queue-Name	Order | order.neworder
Zweck / Semantik	Auf der Applikation soll eine neue Bestellung erfasst werden und damit den ganzen Bestellprozess initiieren. Dabei wird der Kunde nach Mahnungen überprüft und die Preise der Artikel für die Bestellbestätigung abgefragt.
Voraussetzung	- Customer muss im Kundenmanagement existieren und mit der übergebenen ID übereinstimmen. - Artikel müssen im Filiallager existieren. Die Artikel müssen aber nicht zwingend vorhanden sein, da sie noch nachbestellt werden können.

Message Aufbau	- customerID (String): ID des Kunden
	- Array «orderPositionList» mit:
	  ○ articleID (String): ID des gewünschten Artikels
	  ○ count (int): Anzahl des entsprechenden Artikels
	Beispiel:
	{ "customerID" : "bcff3ee2-f9ff-4757-a8e7-07ff4b65fd7d" , "orderPositionList" : [{"articleID" : "765f42cc-c319-41c0-9af6-f7c78cfdff7d" , "count" : 3 }] }

Rückgabe	Der Aufbau der Bestellbestätigung wird in der Beschreibung des Bestellmanagements genauer erläutert. **Beispiel:** {"status": "OK", "message": "Order created successfully! Here is your orderconfirmation: ", "data": "{\"date\":\"2020-05-25@08:46:49\", \"customerID\":\"1\",\"orderID\":\"5ecb85f92794d3148d31551a\", \"customerFirstName\":\"Hans\",\"customerLastName\":

	\"Wurst\",\"wholePrice\":90.00, \"articleList\":[{\"articleID\":\"765f42cc-c319-41c0-9af6-f7c78cfdff7d\",\"count\":3,\"available\":false,\"price\":30.00}],\"_id\":765f42cc-c319-41c0-9af6-f7c78cfdff9d }"}
Protokoll	Synchron
Eigenschaften	Verfügbarkeit: Bestellungen können nur erstellt werden, wenn das Kundenmanagement als auch das Filiallager verfügbar ist.

Tabelle 28: Spezifikation für eine neue Bestellung

Queue-Name	Order | order.getone
Zweck / Semantik	Auf der Applikation soll nach einer Bestellung gesucht und zurückgegeben werden.
Voraussetzung	- Bestell-ID muss gültig sein bzw. existieren
Message Aufbau	ID (String): Hierbei muss lediglich ein einfacher String mit der gesuchten Bestell-ID mitgegeben werden.

Rückgabe	Ein Response-Objekt welches wie folgt aufgebaut ist:
	- Status (OK, ERROR, BAD_REQUEST, PRODUCTS_NOT_AVAILABLE)
	- Message (String): Informationen über die Verarbeitung.
	- Data (String):
	o Bei Erfolg ist darin die Bestellung enthalten.
	o Bei Nichterfolg ist dieses leer.
	Der Aufbau der Bestellung wird in der Beschreibung des Bestellmanagements genauer erläutert.
	Beispiel:
	{
	"status": "OK",
	"message": "Order founded!",
	"data": "{\"date\":\"2020-05-25@08:46:49\",\"state\":\"WAITING_FOR_PRODUCT\",\"customerID\":\"1\",\"orderPositionList\":[{\"articleID\":\"123\",\"count\":3,\"available\":false,\"price\":0.0}],\"_id\":{\"$oid\":\"5ecb85f92794d3148d315519\"}}"
	}

Protokoll	Synchron

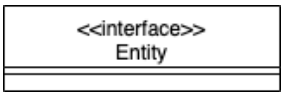
Tabelle 29: Spezifikation, um eine Bestellung zu suchen

Queue-Name	Order | order.getoneconfirmation
Zweck / Semantik	Auf der Applikation soll nach einer Bestellbestätigung gesucht und zurückgegeben werden.
Voraussetzung	- Bestellbestätigungs-ID muss gültig sein bzw. existieren
Message Aufbau	ID (String): Hierbei muss lediglich ein einfacher String mit der gesuchten Bestellbestätigungs-ID mitgegeben werden.
Rückgabe	Ein Response-Objekt welches wie folgt aufgebaut ist: - Status (OK, ERROR, BAD_REQUEST, PRODUCTS_NOT_AVAILABLE) - Message (String): Informationen über die Verarbeitung. - Data (String): o Bei Erfolg ist darin die Bestellbestätigung enthalten. o Bei Nichterfolg ist dieses leer. Der Aufbau der Bestellbestätigung wird in der Beschreibung des Bestellmanagements genauer erläutert. **Beispiel:** `{` `"status": "OK",` `"message": "Order Confirmation founded!",` `"data": "{\"date\":\"2020-05-25@08:46:49\",\"customerID\":\"1\",\"orderID\":\"5ecb85f92794d3148d31551d\",\"customerFirstName\":\"Hans\",\"customerLastName\":\"Wurst\",\"wholePrice\":0.0,\"articleList\":[{\"articleID\":\"123\",\"count\":3,\"available\":false,\"price\":0.0}],\"_id\":\"765f42cc-c319-41c0-9af6-f7c78cfdff9d\" }"` `}`
Protokoll	Synchron

Tabelle 30: Spezifikation, um eine Bestellbestätigung zu suchen

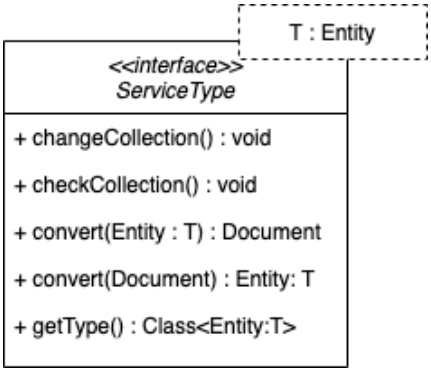
9.4.2. Klassen-APIs

9.4.2.1. Entity



Name	Entity
Zweck / Semantik	Diese Schnittstelle wurde implementiert, um die Services, welche mit der Datenbank interagieren, generisch zu machen. Jedes Objekt, welches in der Datenbank persistiert werden soll, muss dieses Interface implementieren.
Einsatz, Abläufe, Voraussetzungen und Zusicherungen	-
Operationen	-
Fehlerbehandlung	-
Beispiele	-

9.4.2.2. ServiceType



Name	ServiceType
Zweck / Semantik	Definiert eine einfache Schnittstelle, um einen Service für eine Entity zu implementieren, welche man in der Datenbank des Bestellmanagements persistieren bzw. verwalten möchte.
Einsatz, Abläufe, Voraussetzungen und Zusicherungen	- Das Entity, mit dem der Service arbeiten soll, muss das Interface «Entity» implementieren. - Die getType()-methode muss implementiert werden, bevor mittels den Convert-Methoden diese mit dem MongoDbAdapter persistiert werden können. - Vor jeder Interaktion mit dem MongoDbAdapter muss die Methode checkCollection() ausgeführt werden, um sicherzustellen, dass auf der richtigen Collection gearbeitet wird. - Im Konstruktor muss der MongoDbAdapter als auch der Typ der Entitiy mitgegeben werden.

Operationen	Dabei wird auf das JavaDoc verwiesen welches sich im Source-Code befindet.
Fehlerbehandlung	Auftretende Exceptions (Checked-Exceptions) werden an den Aufrufer weitergegeben und sollen nicht hier behandelt werden.
Beispiele	Beispiel einer createEntity-Implementation: `public OrderConfirmation createOrderConfirmation(final OrderConfirmation orderConfirmation) throws JsonProcessingException { orderConfirmation.setId(new ObjectId()); orderConfirmation.setDate(new Date()); this.checkCollection(); this.adapter.create(this.convert(orderConfirmation)); return orderConfirmation; }`  Beispiel eines Konstruktors: `OrderConfirmationService(final MongoDbAdapter adapter, final Class<OrderConfirmation> type) { this.adapter = adapter; this.type = type; }`

10. Epics

Tabelle 31 Enthält die Epics, wie sie in der Aufgabenstellung vorgegeben wurden.

Umgesetzt wurden Epic 1 und 7.

ID	Beschreibung	Prio
1	Das Verkaufspersonal kann Bestellungen im System erfassen, wobei die Artikel ausgewählt, die Verfügbarkeit geprüft, Bestellung ausgeführt, Rechnungen erzeugt und Bestellbestätigungen versendet werden.	A
2	Der Filialleiter kann jederzeit die aktuellen Bestellungen, Nachbestellungen und Lieferungen einsehen.	C
3	Der/die Datentypist/in trägt im System angelieferte Artikel im Filiallager ein.	C
4	Das Verkaufspersonal und der Filialleiter können jederzeit den Zustand der Bestellungen einsehen. Eine Bestellung kann auch geändert oder annulliert werden.	C
5	Auch wenn mehrere Benutzer einer Filiale gleichzeitig im System arbeiten, dürfen keine inkonsistenten Zustände entstehen (Bsp.: gleiches Exemplar mehrfach verkaufen).	B
6	Benutzer/innen des Systems müssen sich mit User-ID und Passwort identifizieren, ihnen werden danach entsprechende Rechte zugeteilt. Benutzergruppen: SysAdmin, Filialleiter/in, Verkäufer/in, Datentypist/in.	B
7	Der Filialverwalter kann die Vorgänge im System (Fehler, wesentliche Geschäftsvorfälle) anhand von Loggingeinträgen laufend überwachen.	B

Tabelle 31: Zusammenfassung der Epics gemäss der Aufgabenstellung

11. User-Stories

ID	Story Title	User Story	Akzeptanzkriterium	Randbedingungen (mit PO definiert)	Prio (1 – 3)	Aufwand (h)
Sprint 1						
1	Kunden erfassen	Als Verkäufer möchte ich einen neuen Kunden erfassen können, um diesem eine Bestellung zuzuweisen.	- Der neue Kunde wird im System erfasst und persistiert. - Folgende Kundendaten werden erfasst: Vorname, Nachname, Strasse, Nummer, PLZ, Ort, Email, Telefon, UUID - Einmalige Kundennummer wird vom System generiert. - Die Kundenerfassung erfolgt synchron.	- Die Zuweisung in die Bestellung ist nicht Bestandteil dieser User Story.	1	4
Sprint 2						
2	Einsicht in das lokale Lager	Als Verkäufer möchte ich alle Artikel im lokalen Lager sehen damit ich deren Verfügbarkeit prüfen und diese einer Bestellung zuweisen kann.	- Übersichtsliste über alle vorhandenen Artikel des lokalen Lagers sind für den Verkäufer jederzeit ersichtlich. - Es sind folgende Attribute eines Artikels ersichtlich: Bezeichnung, ArtikelNummer, Preis, Anzahl (lokal) vorhandener Exemplare. - Die vorhandenen Artikel sind in der Datenbank persistiert. - Die Abfrage erfolgt synchron und dauert < 3 Sekunden.	- Die Zuweisung in die Bestellung ist nicht Bestandteil dieser UserStory - Die Rolle «Verkäufer» wird gefaket da das Usermanagement noch nicht vorhanden ist.	1	4

ID	Story Title	User Story	Akzeptanzkriterium	Randbedingungen (mit PO definiert)	Prio (1 – 3)	Aufwand (h)
3	Bestellung erfassen	Als Verkäufer möchte ich eine Bestellung im Bestellsystem erfassen können damit ich den Bestellprozess initiieren kann.	- Die Bestellung wird im System erfasst und stosst den Bestellungsprozess an. - Es können Artikel aus dem lokalen Lager der Bestellung zugewiesen werden. - Die Information «Anzahl Artikel» wird vom Verkäufer eingegeben. - Wenn eine Anzahl höher als die im lokalen Lager verfügbar eingegeben wird, erscheint eine Meldung. - Die Zahl im Eingabefeld wechselt auf die verfügbare Anzahl. - Es können Kunden, falls sie bereits existieren, der Bestellung zugewiesen werden. - Der Prozess «Kunden erfassen» kann gleichzeitig initiiert und der Bestellung zugewiesen werden. - Bei einer Bestellung wird folgendes erfasst: Artikel (Bezeichnung und Menge), Preis, Verkäufer/-in, Datum/Uhrzeit und Kunde	- Anstoss des Bestellprozesses wird «gemockt». - Die Rolle «Verkäufer» wird gefaket da das Usermanagement noch nicht vorhanden ist.	1	5
Sprint 3						
4	Artikel eintragen	Als Datentypist kann ich einen Artikel im lokalen Lager eintragen damit dieser für die Verkäufer zur Verfügung steht.	- Für den Verkäufer ist der neu eingetragene Artikel innerhalb 10 Sekunden ersichtlich.	- Voraussetzung: Artikel nachbestellen muss erledigt sein. - Die Rollen werden «gemockt», da noch	2	3

ID	Story Title	User Story	Akzeptanzkriterium	Randbedingungen (mit PO definiert)	Prio (1 – 3)	Aufwand (h)
			- Der Datentypist kann nur Artikel eintragen, die im lokalen Lager angekommen sind. - Der Datentypist kann nur Artikel eintragen, die für diese Filiale zum Sortiment gehören. - Es sind folgende Attribute eines Artikels ersichtlich: Bezeichnung, ArtikelNummer, Preis, Anzahl (lokal) vorhandener Exemplare. - ArtikelNummern sind in der Komponente «Zentrallager» definiert und können in der Komponente Bestellsystem nicht verändert werden. - Die Artikelinformationen können im Bestellsystem aus einer Pickliste ausgewählt werden.	kein Usermanagement eingerichtet wurde.		
5	Kunde suchen	Als Verkäufer möchte ich einen Kunden anhand der Nachnamen oder der Kundennummer finden können um ihn einer Bestellung zuweisen zu können.	- Eingabe des Nachnamens gibt eine Liste aller erfassten Kunden mit diesem Namen zurück. - Eingabe der ID gibt den Kunden zurück - Die Abfrage erfolgt synchron und dauert nicht länger als 3 Sekunden.		1	3
6	Liste aller Kunden anzeigen	Als Verkäufer möchte ich eine Liste aller Kunden anzeigen können.	- Wenn Kunden erfasst sind, kommen alle Kunden zurück. - Wenn keine Kunden erfasst sind, kommt eine leere Liste zurück.		1	4

ID	Story Title	User Story	Akzeptanzkriterium	Randbedingungen (mit PO definiert)	Prio (1 – 3)	Aufwand (h)
			- Die abfrage erfolgt synchron und dauert nicht länger als 3 Sekunden.			
6	Artikel automatisch nachbestellen	Als Verkäufer möchte ich im lokalen Lager eine minimale Anzahl Exemplare der Artikel besitzen, um nicht manuell jederzeit nachbestellen zu müssen.	- Bei Unterschreitung der Minimalmenge von einzelnen Artikeln wird automatisch nachbestellt und der Artikel steht für die Eintragung bereit.		1	3
7	Artikel manuell nachbestellen	Als Verkäufer möchte ich für das lokale Lager Artikel nachbestellen können, um Bestellungen zu verarbeiten.	- Nachbestellte Artikel stehen im lokalen Lager für die Eintragung bereit.		1	4
8	Bestellbestätigung erstellen	Als Verkäufer möchte ich bei erfolgreicher Bestellung eine Bestellbestätigung für den Kunden erhalten, damit dieser einen entsprechenden Nachweis für seien Bestellung hat.	- Die Bestellbestätigung erhält man nur bei einer erfolgreich getätigten und eingegangenen Bestellung.	- Bei der Bestellbestätigung handelt es sich um einen Datensatz im JSON-Format, welches folgende Datensätze beinhaltet: Kundendaten und bestellte Artikel.	1	4
9	Monitoring der Geschäftsvorfälle	Als Filialverwalter möchte ich die Vorgänge im System (Fehler, wesentliche Geschäftsvorfälle) anhand von Loggingeinträgen laufend überwachen.	- Als Filialverwalter kann ich jederzeit sehen, wenn Kunden erfasst wurden. - Wenn Fehlermeldungen im System vorkommen. - Wenn Artikel eingetragen werden. - Wenn Bestellungen ausgelöst werden.	-	1	8

ID	Story Title	User Story	Akzeptanzkriterium	Randbedingungen (mit PO definiert)	Prio (1 – 3)	Aufwand (h)
			- Wenn eine Rechnung erstellt wird.			

Tabelle 32: User Stories

12. Testdesign & Abläufe

Die Komponenten werden auf mehreren Levels getestet. Unit-Tests sowie Performance-Tests werden ausschliesslich automatisiert durchgeführt. Integration-Tests können sowohl automatisch als auch manuell durchgeführt werden.

Integration-Tests	Manuelles-Testing
	Automatisiertes-Testing
Performance-Tests	
Unit-Tests	

12.1. Automatisierte Testfälle

Die automatisierten Testfälle befinden sich im Source-Code der git Repositories.

12.2. Manuelle Testfälle

Bei den manuellen Tests geht es vor allem darum, Funktionen zu testen welche systemübergreifende Auswirkungen haben.

Bei den manuellen Tests geht es vor allem darum, Funktionen zu testen, welche systemübergreifende Auswirkungen haben.

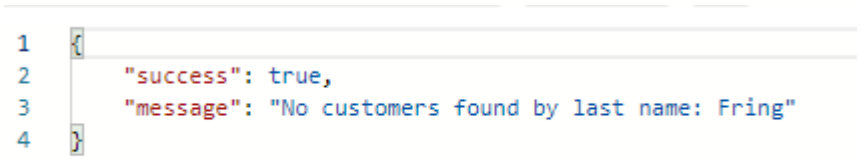
ID	T-01.1	**Version**	1.0.0	**User-Story**	Kunde erfassen
Name	Kunde erfassen				
Beschreibung	Prüft, ob die folgenden Kriterien erfüllt werden: - Der neue Kunde wurde erfasst und in der Datenbank persistiert. - Es wird eine neue Kundennummer erstellt und zusammen mit dem Kunden in der CustomerResponse zurückgegeben. - Logs werden erfasst und angezeigt.				
Voraussetzung	- Keine.				
Ablauf	1. Postman starten. 2. Einen neuen Tab öffnen, POST einstellen und folgende URL eingeben: http://restinterface.appe-g03.el.eee.intern/api/v1/customers/ 3. Im Body erst «raw» und dann JSON wählen. 4. Im Body den unter Anmerkungen hinterlegten Testkunden eingeben. 5. Senden.				

Erwartetes Ergebnis	{
	"customer": {
	"uuid": "164463b4-f11d-446f-a93e-00155d7b4672",
	"first_name": "James",
	"last_name": "McGill",
	"street": "Juan Tabo Blvd. NE",
	"number": "160",
	"zip": "8004",
	"city": "Albuquerque",
	"email": "jesse.pinkman@breakingbad.com",
	"phone": "+0787894504",
	"admonitionlevel": "NOTHING",
	"admonitionLevel": "NOTHING"
	},
	"success": true
	}

Anmerkungen	Testkunde:
	{
	"first_name": "James",
	"last_name": "McGill",
	"street": "Juan Tabo Blvd. NE",
	"number": "160",
	"zip": "8004",
	"city": "Albuquerque",
	"email": "jesse.pinkman@breakingbad.com",
	"phone": "+0787894504"
	}

ID	T-01.2	**Version**	1.0.0	**User-Story**	Kunde erfassen
Name	Kunde per Nachname suchen				
Beschreibung	Prüft, ob die folgenden Kriterien erfüllt werden: - Wenn der Kunde existiert, wird er zurückgegeben. - Wenn der Kunde nicht existiert, wird eine leere Liste zurückgegeben. - Logs werden erfasst und angezeigt.				
Voraussetzung	- Keine.				

Ablauf	6. Postman starten. 7. Einen neuen Tab öffnen, GET einstellen und folgende URL eingeben: restinterface.appe-g03.el.eee.intern/api/v1/customers/?last_name=White 8. Den Body Leer lassen. 9. Senden. 10. In Postman die URL anpassen auf restinterface.appe-g03.el.eee.intern/api/v1/customers/?last_name=Fring 11. senden

Erwartetes Ergebnis	`Nach Schritt 9 kommt eine Liste von Kunden mit dem Name White, aber mit unterschiedlichen UUIDS zurück.`
	`Nach Schritt 11 kommt folgende Meldung:`
	 

Anmerkungen	

ID	T-02	**Version**	1.0.0	**User-Story**	Bestellung erfassen
Name	Von der Kundenerfassung bis zur Rechnung				
Beschreibung	Prüft, ob die folgenden Kriterien erfüllt werden: - Der neue Kunde wurde erfasst und in der Datenbank persistiert. - Der neue Kunde ist über die ID abrufbar. - Es wird eine Liste mit den aktuellen Artikeln angezeigt. - Die neue Bestellung wurde erfasst und in der Datenbank persistiert. - Die neue Bestellung ist über die ID abrufbar. - Alle bestellten Artikel wurden reserviert. - Es wurde eine Rechnung für den Kunden erstellt und persistiert. - Die Rechnung ist über die ID Abrufbar.				
Voraussetzung	- Es müssen genug Artikel im Filiallager vorhanden sein, so dass diese reserviert werden können, um den Rechnungsprozess zu starten.				
Ablauf	12. Es wird ein neuer Kunde erfasst. 13. Die ID des erfassten Kunden wird abgefragt. 14. Es wird eine Liste aller verfügbaren Artikel abgefragt. 15. Es wird eine Bestellung mit dem neu erfassten Kunden und einem Artikel erfasst, von dem genug vorhanden sind.				

	16. In der Bestellbestätigung wird der Gesamtpreis überprüft. 17. Die ID der neu erfassten Bestellung wird abgefragt. 18. Die ID der erstellten Rechnung wird abgefragt und der Preis wird mit der Bestellbestätigung verglichen.
Erwartetes Ergebnis	- Bei der Abfrage der entsprechenden IDs erhalte ich das gewünschte Resultat. - Im Logz.io kann ich jeden einzelnen Schritt anhand der erfassten Logeinträge nachschauen. - Die Rechnung beinhaltet die bestellten Artikel und den korrekten Preis. - Im Filiallager wurde der Artikel entsprechend abgebucht.
Anmerkungen	-

ID	T-03.1	**Version**	1.0.0	**User-Story**	Kunde erfassen, Artikelanzahl aktualisieren, Artikel eintragen
Name	Ausfall vom Orderservice				
Beschreibung	Prüft, ob die folgenden Kriterien erfüllt werden: - Wenn der Orderservice ausfällt, sind trotzdem alle Funktionalitäten des Kundenmanagements als auch des Filiallagers funktionstüchtig.				
Voraussetzung	- Während dem Test darf der Orderservice nicht erreichbar sein.				
Ablauf	1. Der Orderservice wird heruntergefahren bzw. «gekillt». 2. Es wird ein Customer erstellt. 3. Der neuerstellte Customer wird abgefragt. 4. Es werden alle Customer abgefragt. 5. Es werden alle Artikel abgefragt. 6. Es wird ein neuer Artikel hinzugefügt. 7. Es werden nochmals alle Artikel abgefragt.				
Erwartetes Ergebnis	- Der Customer wurde erfolgreich erfasst. - Bei der Abfrage des Customers erhält man den gewünschten Customer. - Bei der Abfrage der Artikel erhält man eine Liste aller verfügbaren Artikel. - Nachdem ein Artikel hinzugefügt wurde, ist der aktualisierte Artikel verfügbar.				

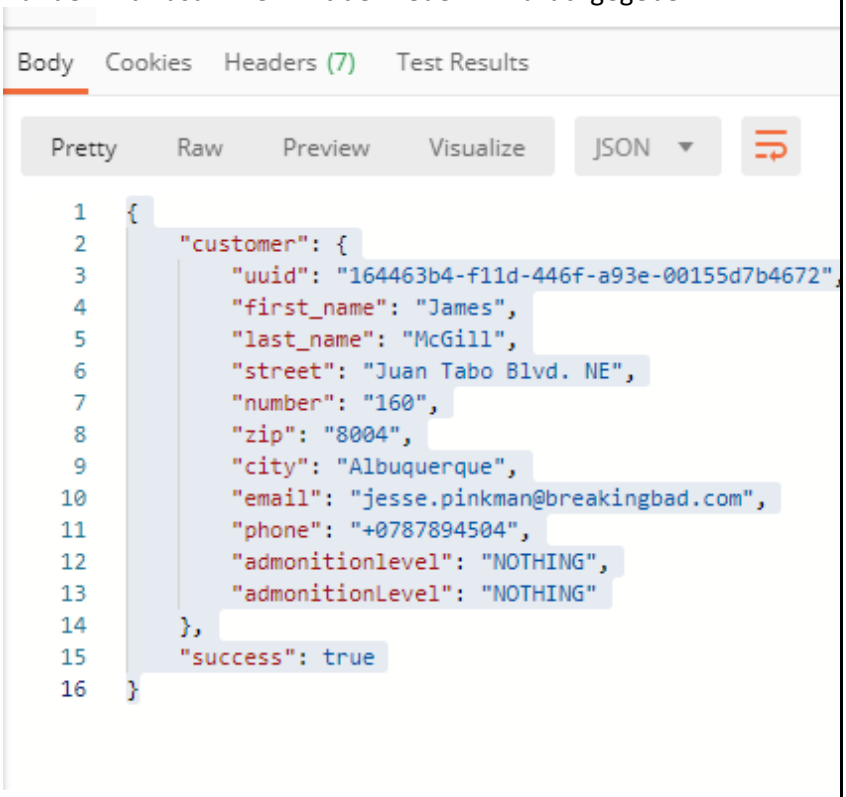
Anmerkungen	-

ID	T-03.2	**Version**	1.0.0	**User-Story**	Bestellung erfassen
Name	Ausfall vom Customerservice				
Beschreibung	Prüft, ob die folgenden Kriterien erfüllt werden: - Wenn der Customerservice ausfällt, sind trotzdem alle Funktionalitäten des Bestellmanagements als auch des Filiallagers funktionstüchtig.				
Voraussetzung	- Während dem Test darf der Customerservice nicht erreichbar sein.				
Ablauf	1. Der Customerservice wird heruntergefahren bzw. «gekillt». 2. Es wird eine Bestellung erstellt. 3. Die neuerstelle Bestellung wird abgefragt. 4. Die neuerstellte Bestellbestätigung wird abgefragt. 5. Es wird eine andere Bestellung bzw. Bestellbestätigung abgefragt. 6. Es werden alle Artikel abgefragt. 7. Die Anzahl eines Artikels wird aktualisiert. 8. Die Artikel werden erneut abgefragt und es wird überprüft, ob die Artikelnummer aktualisiert wurde.				
Erwartetes Ergebnis	- Die Bestellung wurde erfolgreich erfasst. - Bei der Abfrage der Bestellung erhält man die gewünschte Bestellung. - Bei der Abfrage der Bestellbestätigung erhält man die gewünschte Bestellbestätigung. - Bei der Abfrage der Artikel erhält man eine Liste aller verfügbaren Artikel. - Nachdem ein Artikel aktualisiert wurde ist der aktualisierte Artikel verfügbar.				
Anmerkungen	-				

ID	T-03.3	**Version**	1.0.0	**User-Story**	Bestellung erfassen, Kunde erfassen
Name	Ausfall vom Filiallager				
Beschreibung	Prüft, ob die folgenden Kriterien erfüllt werden: - Wenn das Filiallager ausfällt, sind trotzdem alle Funktionalitäten des Bestellmanagements als auch des Kundenmanagements funktionstüchtig.				
Voraussetzung	- Während dem Test darf das Filiallager nicht erreichbar sein.				
Ablauf	1. Das Filiallager wird heruntergefahren bzw. «gekillt». 2. Es wird eine Bestellung erstellt. 3. Die neuerstelle Bestellung wird abgefragt. 4. Die neuerstellte Bestellbestätigung wird abgefragt. 5. Es wird eine andere Bestellung bzw. Bestellbestätigung abgefragt. 6. Es wird ein Customer erstellt. 7. Der neuerstellte Customer wird abgefragt. 8. Es werden alle Customer abgefragt.				
Erwartetes Ergebnis	- Die Bestellung wurde erfolgreich erfasst. - Bei der Abfrage der Bestellung erhält man die gewünschte Bestellung. - Bei der Abfrage der Bestellbestätigung erhält man die gewünschte Bestellbestätigung. - Der Customer wurde erfolgreich erfasst. - Bei der Abfrage des Customers erhält man den gewünschten Customer.				
Anmerkungen	-				

13. Testprotokoll

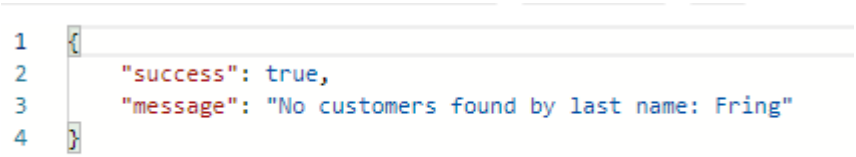
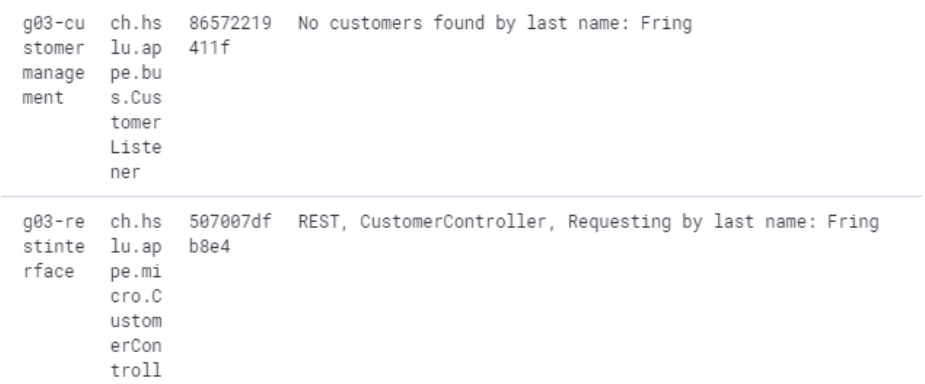
Manuelle Testfälle

Test-ID	T-01.1
Tester	Gabriela Moos
Durchführungsdatum	29.05.2020
Testergebnis	- Kunde «James McGill» wurde erstellt. - Kunde wird zusammen mit der neuen ID zurückgegeben:  - Logs werden erfasst: 

Fehlerbeschreibung	-

Test-ID	T-01.2
Tester	Gabriela Moos
Durchführungsdatum	29.05.2020
Testergebnis	- Nach Schritt 9 wird eine Liste der Kunden mit dem Namen «White» zusammen zurückgegeben:


```
{
    "customers": [
        {
            "uuid": "1f15aacc-9e4b-4e73-85db-8943dc2ea18c",
            "first_name": "Walter",
            "last_name": "White",
            "street": "Wyoming Bldv",
            "number": "1501",
            "zip": "8002",
            "city": "Albuquerque",
            "email": "walter.white@breakingbad.com",
            "phone": "+41787894502",
            "admonitionlevel": "FIRST_LEVEL",
            "admonitionLevel": "FIRST_LEVEL"
        },
        {
            "uuid": "8be6e74f-f97f-421a-8f5c-2bde2539cc6f",
            "first_name": "Walter",
            "last_name": "White",
            "street": "Wyoming Bldv",
            "number": "1501",
            "zip": "8001",
            "city": "Albuquerque",
            "email": "walter.white@breakingbad.com",
            "phone": "+41787894502",
            "admonitionlevel": "NOTHING",
            "admonitionLevel": "NOTHING"
        },
        {
            "uuid": "e8511aca-673d-4896-a199-5c911b68b19e",
            "first_name": "Walter",
            "last_name": "White",
            "street": "Wyoming Bldv",
            "number": "1501",
            "zip": "8001",
            "city": "Albuquerque",
            "email": "walter.white@breakingbad.com",
```

	- Logs für gefundene Kunden mit Namen White werden erfasst:  Nach Schritt 11 wird die erwartete Meldung angezeigt:  Die Logs zeigen, dass der Kunde gesucht, aber nicht gefunden wurde: 
Fehlerbeschreibung	-

Test-ID	T-02
Tester	Frederico Fischer
Durchführungsdatum	25.05.2020
Testergebnis	- Kunde «Frederico Fischer» wurde erstellt. - Kunde «Frederico Fischer» ist über ID 7b238395-b905-4cb0-b0d5-

	6bc6133b3488 abrufbar - Log-Einträge für erstellen Kunden wurden gemacht:  - Bestellung wurde erfasst. Bestellbestätigung wurde zugestellt. - Bestellung ist über ID «5ecbbe074d75d712af925447» abrufbar - Artikel wurden im Filiallager abgebucht - Es wurde eine Rechnung erstellt. - Die Rechnung ist über die ID «5ecbbe07478e7a67b9dd8244» abrufbar. - Log-Einträge für erfasste Bestellung und Rechnung wurden gemacht: 
Fehlerbeschreibung	-

Test-ID	T-02
Tester	Frederico Fischer
Durchführungsdatum	25.05.2020
Testergebnis	- Kunde «Frederico Fischer» wurde erstellt. - Kunde «Frederico Fischer» ist über ID 7b238395-b905-4cb0-b0d5-6bc6133b3488 abrufbar - Log-Einträge für erstellen Kunden wurden gemacht:  - Bestellung wurde erfasst. Bestellbestätigung wurde zugestellt. - Bestellung ist über ID «5ecbbe074d75d712af925447» abrufbar - Artikel wurden im Filiallager abgebucht - Es wurde eine Rechnung erstellt.

	- Die Rechnung ist über die ID «5ecbbe07478e7a67b9dd8244» abrufbar. - Log-Einträge für erfasste Bestellung und Rechnung wurden gemacht: 
Fehlerbeschreibung	-

Test-ID	T-03.1
Tester	Frederico Fischer
Durchführungsdatum	25.05.2020
Testergebnis	- Es konnte ein Customer erstellt werden. - Der neuerstellte Customer konnte erfolgreich abgefragt werden. - Alle Artikel konnten abgefragt werden. - Es konnte ein Artikel hinzugefügt werden.
Fehlerbeschreibung	-

Test-ID	T-03.2
Tester	Frederico Fischer
Durchführungsdatum	25.05.2020
Testergebnis	- Es konnte keine Bestellung erfasst werden. - Die neuerstellte Bestellung wurde im OrderService nicht gefunden. - Die Bestellbestätigung wurde nicht erfasst. - Es konnten alle Artikel abgefragt werden. - Der neuhinzugefügte Artikel konnte abgefragt werden.
Fehlerbeschreibung	Die Bestellung konnte nicht erfasst werden, weil der Customerservice nicht erreichbar war. Der Grund hierfür ist ein Architekturfehler. Die Anfragen vom OrderService zum Customerservice wurden synchron implementiert damit der Verkäufer sofort eine Bestellbestätigung erhält. Dieser Fehler könnte behoben werden, in dem lediglich die BestellID an den Verkäufer zurückgegeben wird und anschliessend über diese die Bestellbestätigung etc. abgefragt werden könnte.

Test-ID	T-03.3
Tester	Frederico Fischer
Durchführungsdatum	25.05.2020
Testergebnis	- Es konnte keine Bestellung erfasst werden. - Die neuerfasste Bestellung konnte nicht abgefragt werden, weil die Bestellung nicht erfolgreich erfasst wurde. - Es konnten alle Bestellungen abgefragt werden. - Es konnten alle Customer abgefragt werden. - Es konnte ein neuer Customer erfasst werden.
Fehlerbeschreibung	Der Ausfall des Filiallagers führt dazu, dass keine neuen Bestellungen erfasst werden können. Darauf folgt, dass auch keine Bestellbestätigung erstellt wird. Der Grund hierfür ist ein Architekturfehler. Die Anfragen vom OrderService zum Filiallager wurden snchron implementiert damit der Verkäufer sofort eine Bestellbestätigung erhält. Dieser Fehler könnte behoben werden, in dem lediglich die BestellID an den Verkäufer zurückgegeben wird und anschliessend über diese die Bestellbestätigung etc. abgefragt werden könnte.