**SuperDuperMarktApp**

Projekt-Dokumentation

Bearbeitet von: Christopher Ridder

Letzte Aktualisierung: 12.12.23

# Inhalt

[Inhalt 2](#_Toc153289948)

[Anforderungen 3](#_Toc153289949)

[Annahmen 3](#_Toc153289950)

[Verwendete Pattern 3](#_Toc153289951)

[Eingereichte Versionen 4](#_Toc153289952)

[Version in Repo ‚SDM\_Monolith‘ 4](#_Toc153289953)

[Version in Repo ‚SuperDuperMarktApp‘ 4](#_Toc153289954)

# Anforderungen

* Die App soll Regalpflegenden die Erfassung von eingeräumten Produkten ermöglichen, sodass jederzeit der Regalbestand eingesehen werden kann.
* Die App soll Regalpflegende dabei unterstützen, die Produkte mit den korrekten Preisen auszuzeichnen, indem produktspezifische und tagesaktuelle Preise berechnet und abgerufen werden können.
* Die App soll Regalpflegende dabei unterstützen, Produkte bei Erreichen ihres Verfallsdatums aus den Regalen zu nehmen, sodass sichergestellt ist, dass keine abgelaufene Ware in den Regalen liegt.
* Die App soll Regalpflegende dabei unterstützen, Produkte bei Unterschreitung eines gegebenen produktspezifischen Qualitätsniveaus aus den Regalen zu nehmen, sodass sichergestellt ist, dass nur Produkte der vorgegebenen Qualität in den Regalen liegen.
* Die App soll Regalpflegenden auf Abruf eine Ansicht bieten, die zu allen Produkten neben dem Namen auch tagesaktuelle Angaben zum Bestand, der Qualität, dem Preis und die Haltbarkeit enthält, wobei zu entsorgende Produkte speziell markiert werden.
* Die App soll die Aufnahme weiterer Produkt-Typen ermöglichen, ohne dass dabei der ‚zentrale Steuerungscode‘ geändert werden muss, sodass eine Aufnahme im laufenden Betrieb möglich ist.
* Die App soll das Einlesen von Produkten via csv ermöglichen, sodass eine beliebige Menge an Produkten auf einmal erfasst werden kann.
* Die App soll das Einlesen von Produkten via SQL ermöglichen, sodass eine beliebige Menge an Produkten auf einmal erfasst werden kann.

# Annahmen

* Es existiert keine Lagerhaltung. Produkte werden direkt in die Regale geräumt. Konsequenz: keine Unterscheidung verschiedener Bestände, wie z.B. Lagerbestand, Regalbestand oder Gesamtbestand als Summe, sondern nur den einen Bestand des Sortiments.
* Im Betrachtungszeitraum finden keine Lieferungen von Produkten statt. Konsequenzen:
  + Der Bestand kann nur kleiner werden.
  + Die Berechnung der ‚Tage seit Lieferung‘, die z.B. für die Berechnung der Tages-Qualität erforderlich sind, ist für alle Produkte gleich, da alle Produkte des Sortiments an nur einem Tag geliefert wurden. Die Berechnung findet daher zentral statt und ist nicht Teil der Produkt-Klasse oder der aus ihr abgeleiteten Klassen.

# Verwendete Pattern

* **Factory Method** [eigene Implementierung]**:** Verschiedene Produkte können mit einer zentralen Erzeuger-Methode generiert werden. Diese wird von speziellen Erzeuger-Klassen überschrieben, die jeweils den spezifischen Produkttyp zurückgeben. Dies ermöglicht das flexible Hinzufügen weiterer Produkt-Typen ohne Änderungen am bestehenden Code.
* **Singleton** [Spring-Framework]**:** Erzeugung einer zentralen Instanz einer Klasse, die an verschiedenen Stellen wiederverwendet wird. Keine eigene Implementierung des Patterns, sondern Nutzung der Spring-Implementierung, also Springs IoC-Container und Dependency Injection z.B. via Autowring oder Constructor Injection von Beans.
* **Builder** [Lombok @Builder Annotation]**:** Die Konstruktion eines Objektswird an einen Builder übergeben. Mit Hilfe des Builders kann das Objekt sukzessive konstruiert werden. Es ist möglich nur die Eigenschaften zu definieren, die erforderlich sind. Eine Definition mehrerer Konstruktoren mit unterschiedlichen Parametern entfällt bei Verwendung des Builder patterns. Nutzung des Patterns über Lomboks @Builder-Annotation.

# Eingereichte Versionen

Es werden zwei Versionen eingereicht, die sich in folgenden Punkten unterscheiden:

* Technische Implementierung
* Handhabung verfallener bzw. qualitätsunterschreitender Produkte
* Enthaltene Produkt-Typen
* Test-Fälle

## Version in Repo ‚SDM\_Monolith‘

Alle Services zur Verwaltung der verschiedenen Produkte bzw. Produkttypen sind in einer monolithischen Applikation zusammengefasst. Die Datenerhaltung erfolgt zentral in einer DB, die die verschiedenen Produkttypen persistiert.

Verfallene oder qualitätsunterschreitende Produkte werden in dieser Version von Regalpflegenden aus den Regalen genommen, sobald dies erforderlich ist. Sie sind in den Übersichten der folgenden Tage folglich nicht mehr enthalten.

Die Version enthält als weiteren ProduktTyp ‚Brot‘. Regeln:

* Brot ändert seine Qualität nicht.
* Brot mit einer Qualität > 0 wird in Regale geräumt.
* Brot hat einen tagesaktuellen Preis. Der reguläre Preis ist an allen Tagen gleich bis auf den letzten Tag vor Erreichen des Verfallsdatums. An diesem Tag wird Brot zur Hälfte des regulären Preises verkauft.
* Der reguläre Preis wird berechnet über den Grundpreis + 0,1\*aktuelleQualität (wobei die aktuelleQualität stets der startQualität entspricht)

In der Klasse ‚ProduktServicesTest‘ sind einige Integration-Tests implementiert.

## Version in Repo ‚SuperDuperMarktApp‘

Die Applikation ist als Microservices Architecture implementiert. Sie besteht aus einer ‘MainApp’, die Services des Weinservice und des Käseservice nutzt. Alle drei Komponenten sind als eigenständige Spring Boots Applikationen implementiert. Die zentrale Steuerung erfolgt über die MainApp. Sie ist auch für die Generierung der Ausgabe verantwortlich. Für die Ausgabe können verschiedene Berichtstypen verwendet werden. Die Wahl des auszugebenden Berichtstyps erfolgt über die Konsole durch den Benutzer.

Die Datenhaltung ist dezentral. Weinservice und Käseservice nutzen je eine eigene DB. Die MainApp verfügt über keine eigene Datenhaltung, sondern verarbeitet ausschließlich Daten, die vom Wein- und/oder Käseservice über REST-APIs bereitgestellt werden.

Um die verschiedenen Komponenten im Kontext des Datenaustauschs auffindbar zu machen, wird der Eureka Discovery Service genutzt. Die drei genannten Komponenten registrieren sich bei dem Eureka-Server, der ebenfalls als eigenständige Applikation läuft, und fragen die Registry des Servers an, um Adressen anderer Komponenten zu erhalten.

Verfallene oder qualitätsunterschreitende Produkte werden in dieser Version nicht eliminiert, sondern ausschließlich als diese gekennzeichnet. Entsprechend sind diese Produkte auch nach Verfall / Qualitätsunterschreitung weiter in den Tages-Übersichten enthalten, sind aber als zu eliminierend gekennzeichnet. Die Entfernung der relevanten Produkte durch die Mitarbeiter wird in dieser Version bewusst nicht simuliert.

In dieser Version ist der Produkt-Typ ‚Brot‘ nicht implementiert und es wurden keine Test-Fälle angelegt.

Um die SuperDuperMarktApp zu starten, müssen die einzelnen Services in folgender Reihenfolge gestartet werden:

1. Eureka-Server
2. WeinService
3. KaeseService
4. MainApp