Das Projekt Weinregal

Inhaltsverzeichnis

Anforderungen

Anwendungsfälle (Use-Cases):

1. Einlagern einer Weinflasche
    a. Flasche auf dem Tisch platzieren
    b. Roboter steuern
    c. Etikett einscannen
    d. Flasche in Katalog-Datenbank aufnehmen
    e. Flasche im Regal ablegen
    f. Flasche in der Regal-Datenbank speichern
2. Auslagern einer Weinflasche
    a. Suchen der Flasche in der Regal-DB
    b. Roboter steuern
    c. Flasche aus Regal entnehmen
    d. Flasche aus Regal-Datenbank entfernen
    e. Flasche vom Tisch nehmen
3. Drehen einer Weinflasche (automatisch)
    a. Suchen zu drehender Flasche in der Regal-DB

b. Roboter steuern
c. Flasche im Regal drehen
d. Datum der letzten Drehung in der Regal-Datenbank aktualisieren

APIs

Die Server-Dienste unterteilen sich in drei Teile mit jeweils eigener API:

Steuerung des Roboters

- Weinflasche (physisch) ins Regal legen
  - Etikett-Scan (Teil des Einlagerns, aber separat aufrufbar)
- Weinflasche dem Regal entnehmen
- Weinflasche im Regal drehen

Zusätzlich umfassen zwei weitere APIs die Verwaltung der Datenbanken für den Weinkatalog und das Weinregal.

Datenbank Weinkatalog

Ein Datensatz im Weinkatalog besteht aus den folgenden Informationen:

`(Wein-ID, Name, Jahrgang, Herkunft, Rebsorte, ⋯, Etikett)`

- Suchen im Bestand mit Filtern
- Katalogeintrag hinzufügen
- Katalogeintrag ändern
- Katalogeintrag löschen

Datenbank Weinregal

Ein Datensatz im Weinregal besteht aus den folgenden Informationen: `(Fachnummer, Flaschen-ID, Wein-ID, DatumEinlagerung, DatumLetzteDrehung, ⋯)`

- Suchen und filtern nach bestimmten Flaschen
- Daten einer Flasche einfügen
- Daten einer Flasche ändern
- Daten einer Flasche entfernen

Architektur

Die Umsetzung der drei APIs erfolgt mit REST. Sie sind zunächst unabhängig voneinander und können auch separat betrieben (und getestet) werden.

Orchestrieung der APIs

Häufig wird die Aufgabe, eine Folge von Operationen aus verschiedenen APIs zu koordinieren, der Client-Applikation überlassen. Alternativ könnten diese APIs jedoch bei Bedarf in einem gemeinsamen Server-Dienst (z. B. einer AxonIQ-Saga) zusammengefasst werden, um die Client-Kommunikation zu vereinfachen. AxonIQ kann dabei die Kommunikation zwischen den einzelnen APIs übernehmen und die Transaktionen koordinieren – selbst bei manuellen Zwischenschritten. In einem solchen Fall könnte eine zusätzliche REST-API bereitgestellt werden (derzeit nicht geplant), die die Orchestrierung der anderen APIs mittels Saga übernimmt.

Frameworks und Technologien

Die Implementierung erfolgt mit dem AxonIQ-Framework basierend auf mit Spring Boot mit der Programmiersprache Kotlin. Als Datenbank wird PostgreSQL verwendet.

Betrieb

Der Server und die Datenbank laufen in Docker-Containern. Die Kommunikation erfolgt in der Regel im JSON-Format. Bei Bedarf kann das System auch in einer Kubernetes-Umgebung betrieben werden.

Dank des Einsatzes von AxonIQ wird die Implementierung von CQRS und Event-Sourcing unterstützt. Jede Aktion wird automatisch als Event gespeichert, wodurch ein Event-Log entsteht, das zur Nachvollziehbarkeit und Fehlerbehebung genutzt werden kann.

Hardware-Voraussetzung

- **RAM:** 8 GB (16 GB für Kubernetes)
- **CPU:** 4 Kerne
- **Speicherplatz:** 100 GB Festplatte

Implementierungsdetails

Im Folgenden wird ausschließlich der erste Use-Case, „Einlagern einer Weinflasche“, detailliert beschrieben. Die weiteren Use-Cases werden entsprechend ausgearbeitet, sobald die Umsetzungsstrategie abgestimmt und eindeutig definiert ist.

Weinflasche einlagern

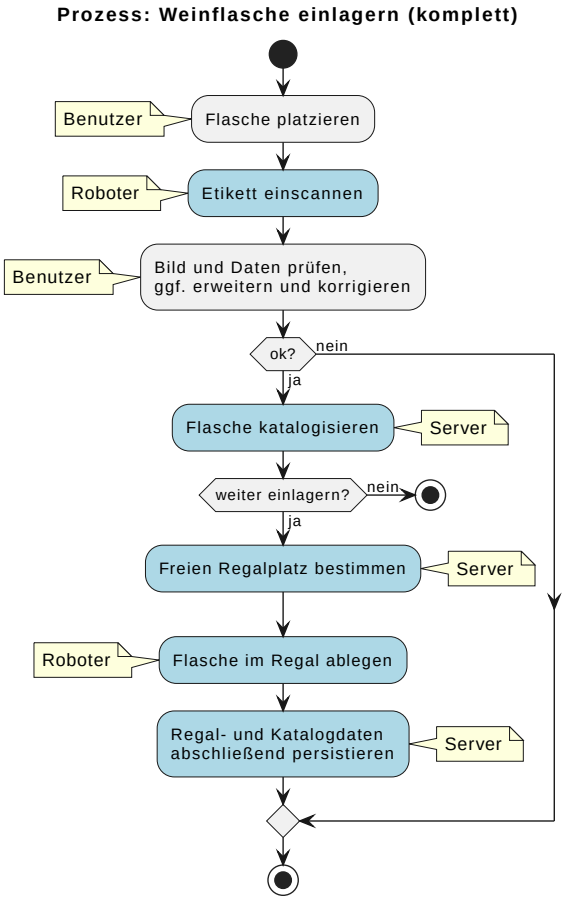
Use Case

- Ziel: Der Benutzer möchte eine Flasche einlagern.
- Akteure
  - Benutzer

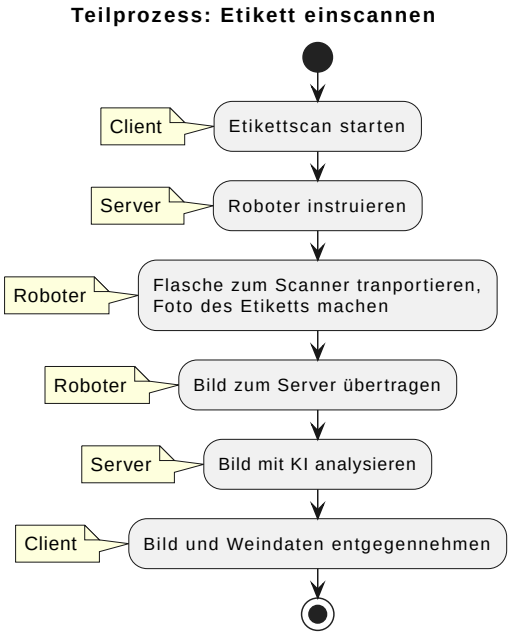
- Server
- Roboter

Gesamter Prozess

Hier wird der Prozess des Einlagerns einer Weinflasche beschrieben. Die Beschreibung erfolgt in Form von Prozessschritten (hier in blau), die im Folgenden jeweils in einem eigenen Sequenzdiagramm dargestellt werden.



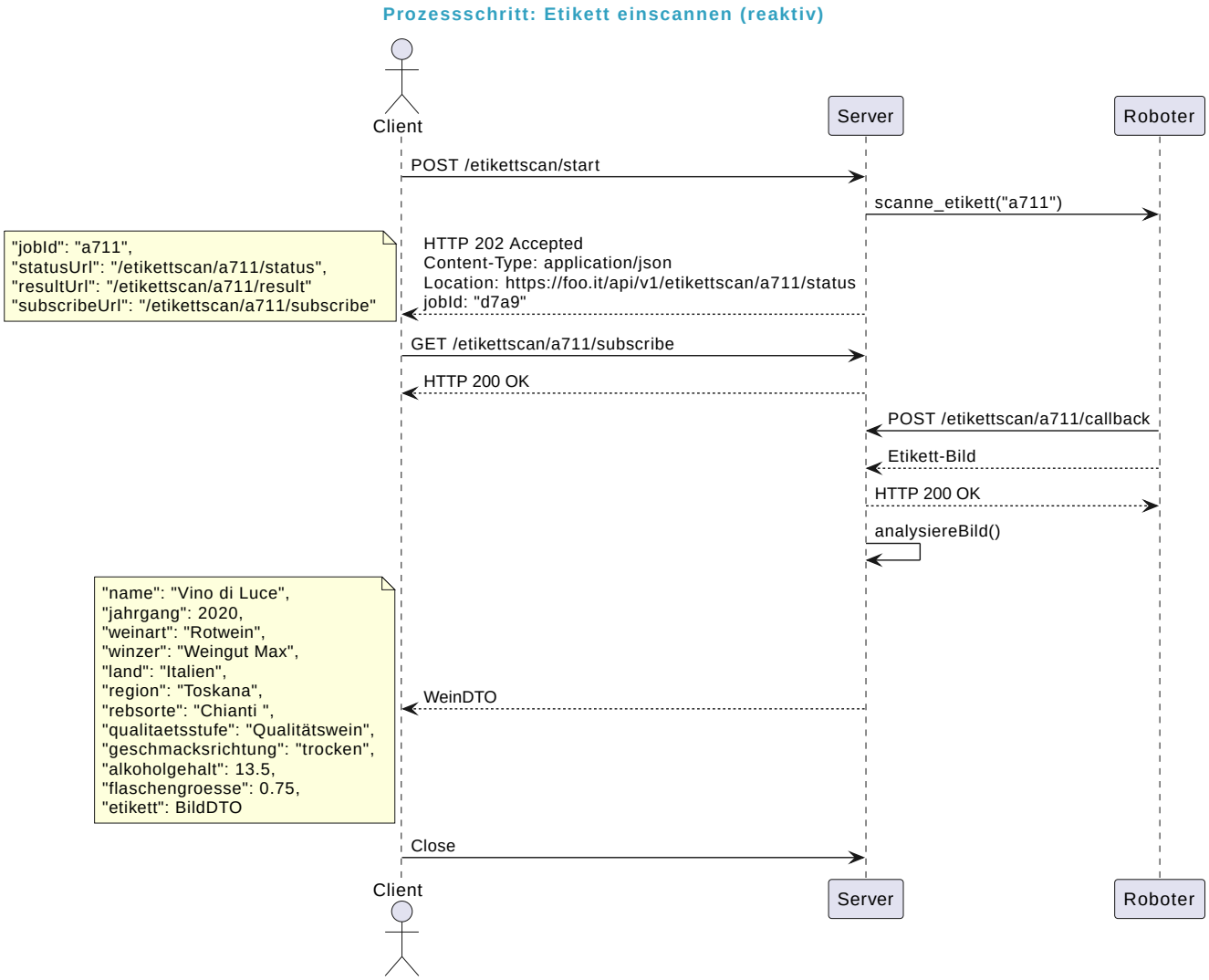
Teilprozess: Etikett einscannen



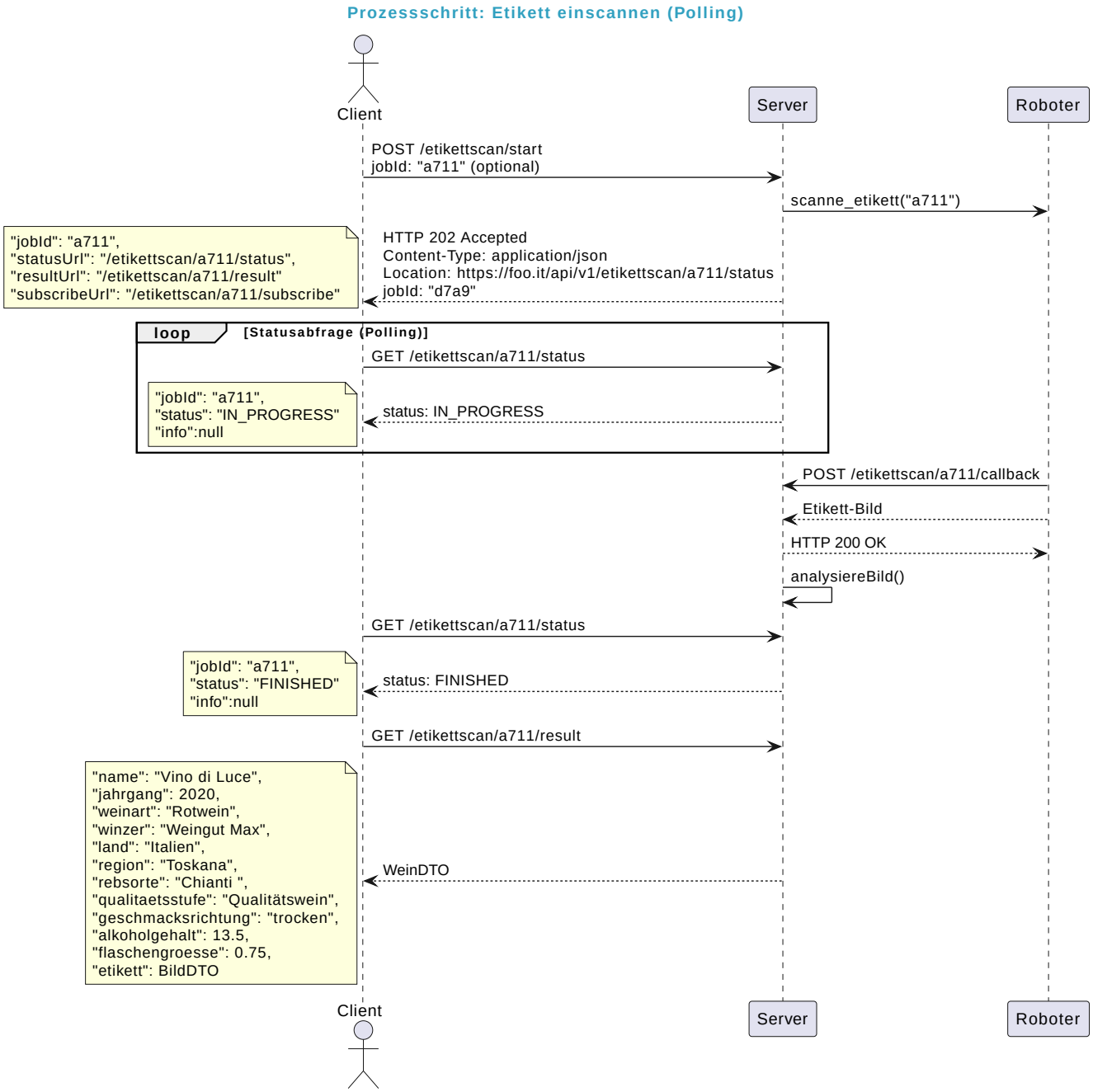
In den folgenden Sequenzdiagrammen werden zwei alternative Möglichkeiten für die asynchrone Verarbeitung des Scan-Vorgangs gezeigt:

1. SSE: der Client registriert sich am Server, um mittels Server-Sent-Event aktiv benachrichtigt zu werden.
2. Polling: Der Client ruft in regelmäßigen Abständen den Status ab; beim Status FINISHED kann das Ergebnis geladen werden.

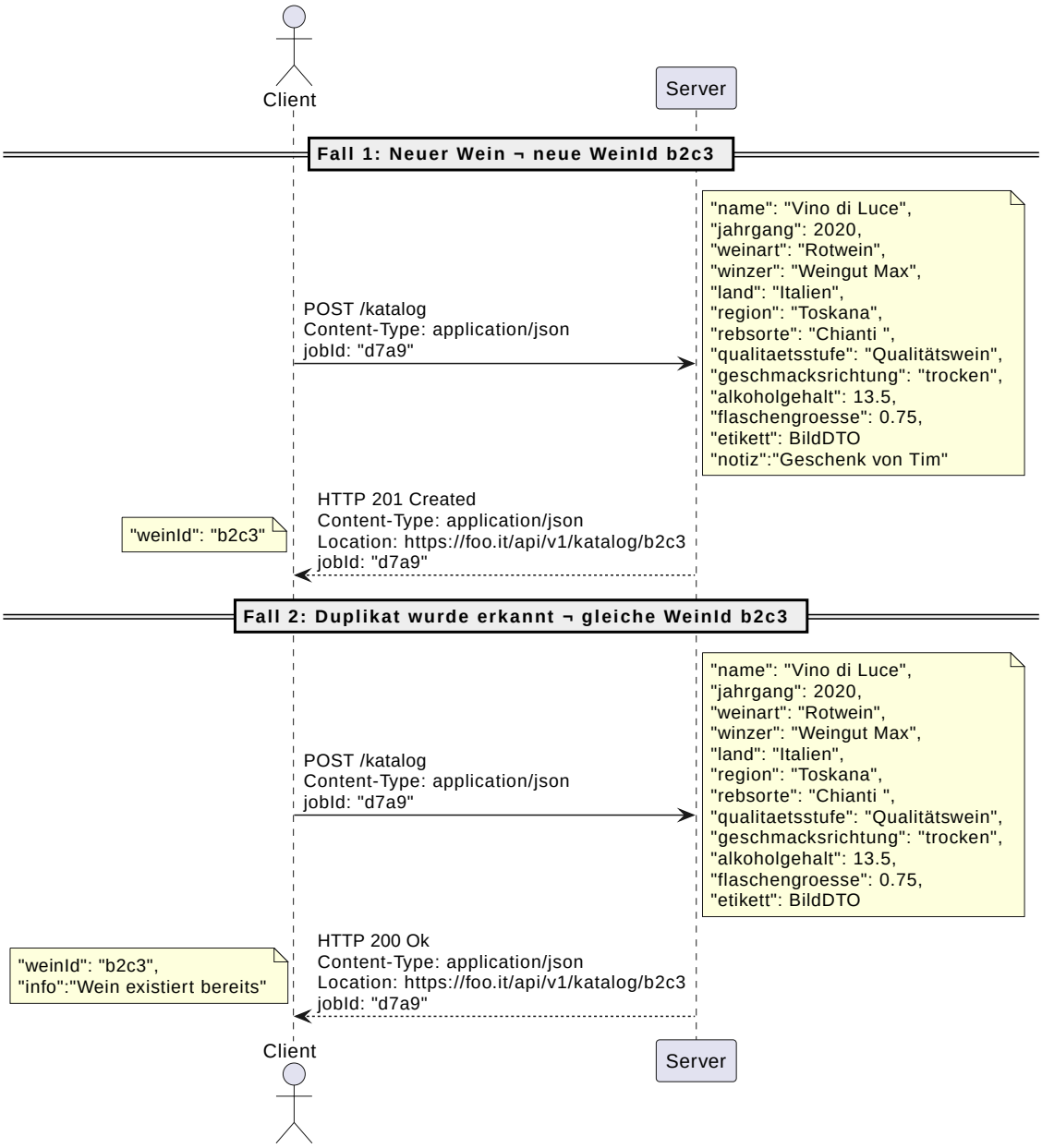
Alternative 1: Mit Server-Sent-Event



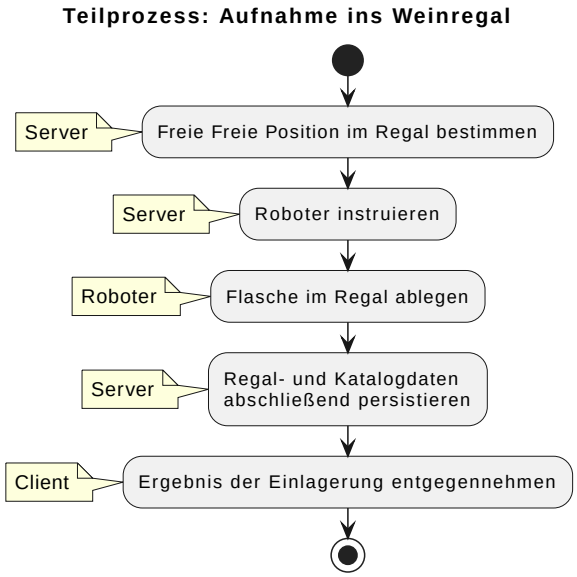
Alternative 2: Mit Polling



Prozessschritt: Flasche katalogisieren

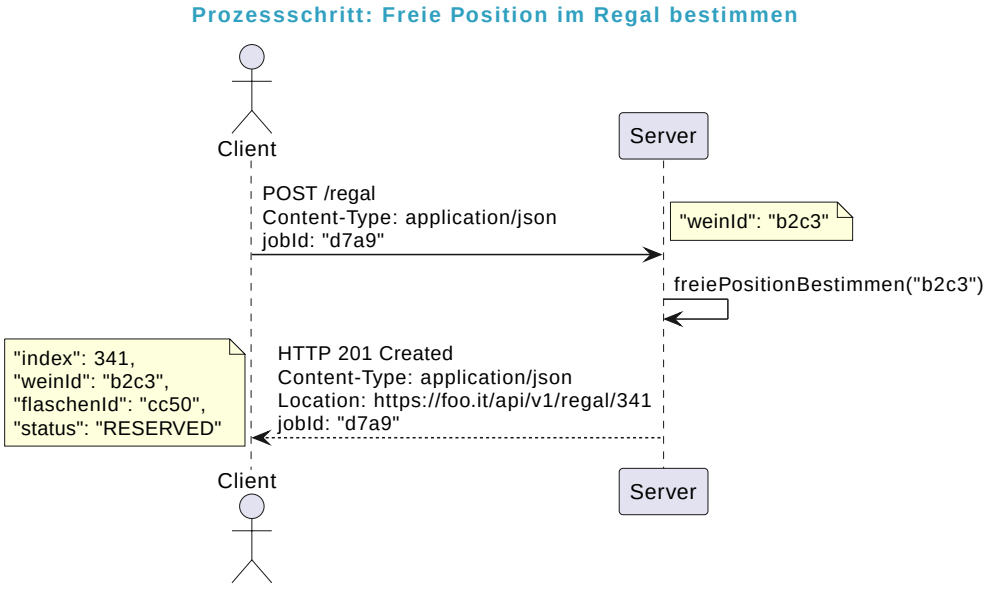


Teilprozess: Aufnehmen ins Weinregal

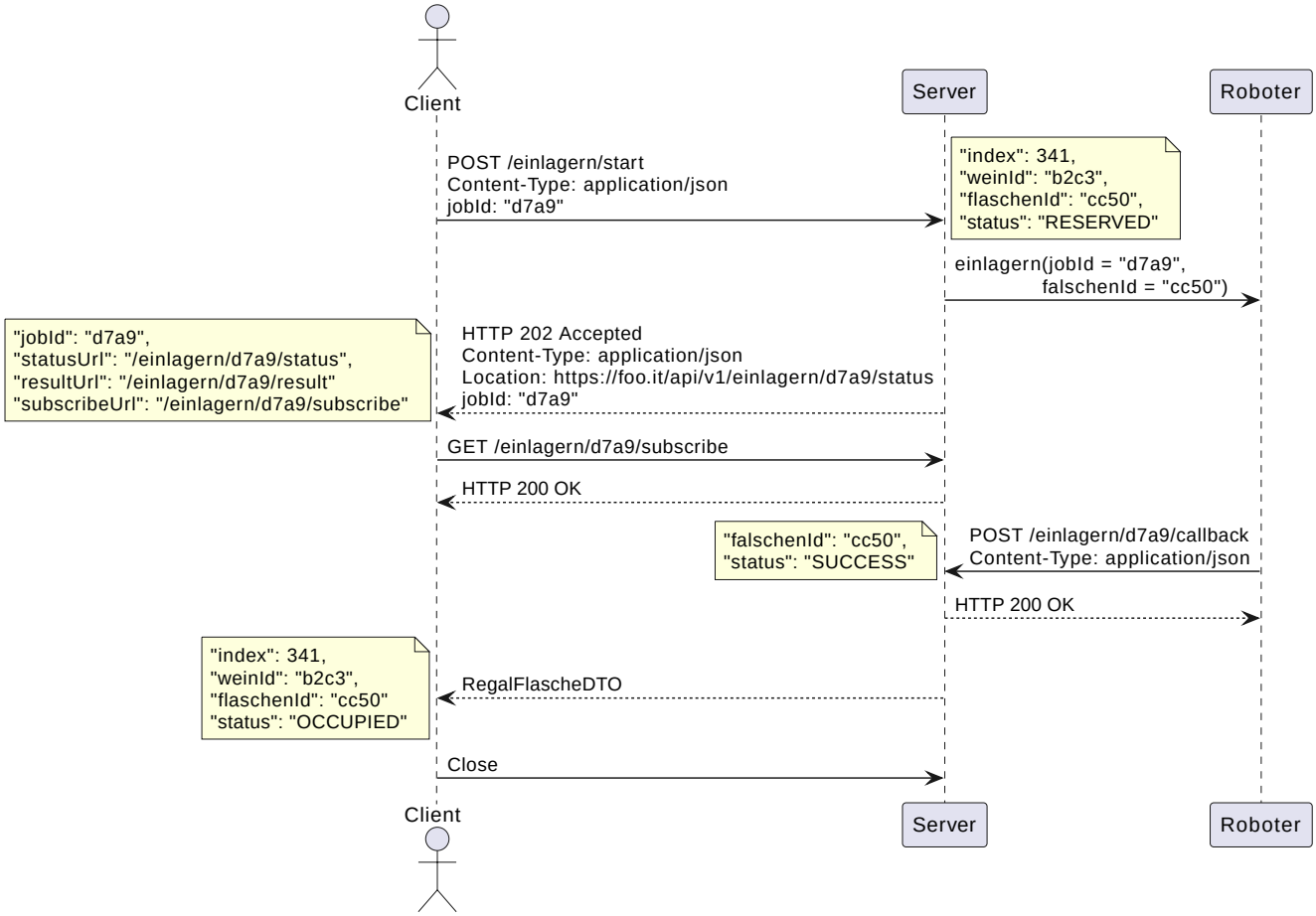


Eventuell kann es vorkommen, dass für die Berechnung der freien Position Informationen über den Wein benötigt werden. Diese könnten

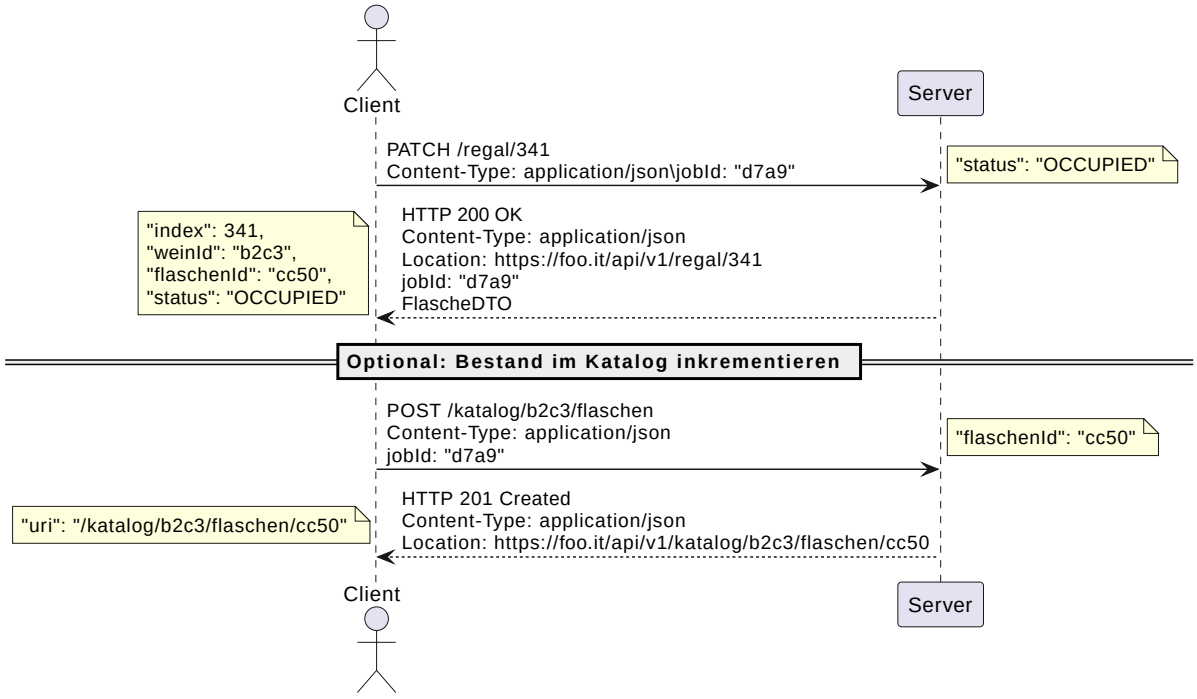
1. direkt beim Aufruf mitgegeben werden (KatalogFlascheDTO)
2. oder später nachgeladen werden (anhand der weinId).



Prozessschritt: Flasche im Regal ablegen



Prozessschritt: Regal- und Katalogdaten abschließend persistieren



Glossar

Im Folgenden wird häufig das englische Vokabular aus der **UML** und dem **Domain-Driven-Design** verwendet:

Use Case

Ein Use Case beschreibt die Interaktion eines oder mehrerer Akteure mit einem System, um ein spezifisches Ziel zu erreichen.

Prozess

Ein Prozess ist eine Abfolge von Aktivitäten, die ein Ziel erreicht. Er beschreibt den Ablauf sowohl auf technischer als auch auf organisatorischer Ebene.

Prozessschritt

Ein Prozessschritt ist ein abgrenzbarer Abschnitt eines Prozesses, der eigenständig bearbeitet werden kann, aber Teil des gesamten Prozesses bleibt.

Client

I.d.R. eine Frontend-Applikation, die Befehle vom Benutzer entgegennimmt und an den Server weiterleitet.

Server

Ein Server ist ein System oder eine Anwendung, die Dienste oder Daten für andere Systeme (Clients) bereitstellt.

Command

Ein Command ist eine Aufforderung an ein System, eine bestimmte Aktion auszuführen und richtet sich stets an genau ein Aggregate.

- Es stammt üblicherweise von dem Client basierend auf einer Benutzereingabe.
- Es löst typischerweise eine Zustandsänderung in dem entsprechenden Aggregate aus.
- In unserem Fall wird das Command mittels REST an den Server gesendet und weiter zu dem zugeordneten Aggregate weitergeleitet.

Event

Ein Event ist eine Benachrichtigung darüber, dass ein bestimmter Zustand oder eine Aktion in der Domäne erfolgt ist. Es signalisiert, dass etwas bereits geschehen ist.

- Ein Command löst in der Regel genau ein Event aus, auf das sich andere Dienste registrieren können, um entsprechend darauf zu reagieren.
- Das gleiche Command kann allerdings unterschiedliche Events auslösen. Beispielsweise kann die Verarbeitung eines Commands erfolgreich oder fehlerhaft verlaufen, was zu zwei verschiedenen Events führt.

Aggregate

Ein Aggregate ist eine logische Gruppierung von Domänenobjekten, die zusammen konsistent gehalten werden. Es stellt die Basiseinheit für Änderungen in der Domäne dar.