1. **Welche Topics nutzen wir für unsere Events?**

**Accounting-Service:**

|  |  |  |
| --- | --- | --- |
| **Event-Name** | **Trigger-Methode** | **Key** |
| accounting.payments.created | POST /payment/create | userId |
| accounting.payments.received | POST /payment/notify/received | userId |
| accounting.payments.statusUpdated | POST /payment/markPaid/{paymentId} | userId |
| accounting.payments.deleted | POST /payment/deleted/{paymentId} | paymentId |

**Term-Service:**

|  |  |  |
| --- | --- | --- |
| **Event-Name** | **Trigger-Methode** | **Key** |
| term.created | POST /term/create | termId |
| term.updated | PUT /term/update/{termId} | termId |
| term.deleted | DELETE /term/delete/{termId} | termId |
| term.caregiver.assigned | POST /term/assign-caregiver | termId |
| term.caregiver.unassigned | POST /term/unassign-caregiver/{termId} | termId |

**2. Schema für Topic-Namen**

Wir verwenden das Schema:

<domain>.<entity>.<event>

Dieses Schema erfüllt folgende Ziele:

* Eindeutigkeit (klar zuordenbar zu Service und Funktion)
* Klare Trennung nach fachlichem Kontext (z. B. accounting.\*, term.\*)
* Einfach erweiterbar für neue Entities und Events

**3. Keys**

**Accounting-Service:**

Bei allen Events, die einen Nutzer betreffen, nutzen wir den userId als Key. Dadurch landen alle Events eines Nutzers in der gleichen Kafka-Partition. Das garantiert:

* korrekte Reihenfolge (z. B. „Zahlung erstellt“ vor „Zahlung erhalten“),
* leichtere Verarbeitung auf Consumer-Seite.

Für Events, die eine konkrete Zahlung betreffen, aber keinen bestimmten Nutzer (z. B. bei Löschung), verwenden wir paymentId als Key. Das macht das Event eindeutig referenzierbar und sinnvoll partitionierbar.

**Term-Service:**

Alle Events betreffen eindeutig einen bestimmten Term. Mit termId als Key wird sichergestellt, dass Events zum selben Termin in derselben Partition landen, was für korrekte Verarbeitung entscheidend ist.