

# CQRS und Event-Sourcing

Von Christian Koch und Jan Burtscher

Command Query Responsibility Segregation, kurz CQRS ist ein bekanntes Pattern zum Datenmanagement. Mit CQRS lassen sich große skalierbare Systeme aufbauen, welche weit aus mehr Schreibvorgänge haben als Lesevorgänge. Dabei gibt es drei wesentliche Datenquellen:

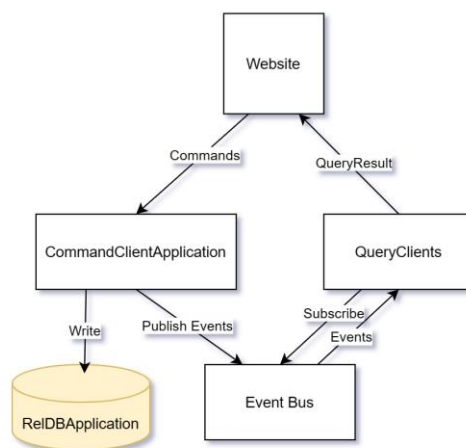
1. Write-Repository
2. Event-Log
3. Read-Repository

Im Write-Repository befindet sich immer der wahre Zustand des Systems. Mit jedem erfolgreichen Schreibvorgang wird ein Event ausgelöst, welches im Event-Log gespeichert wird und über den Event-Bus an die verschiedenen Read-Repositories verteilt wird. So weisen die Read-Repositories eine Eventual Consistency auf. Dies bedeutet, die Daten werden mit einer kleinen Verzögerung aktualisiert. Danach können die Consumer-Anwendungen die perfekt zugeschnittenen Daten für den jeweiligen Anwendungsfall von den Read-Repositories abfragen.

Das vorliegende Gradle-Projekt zeigt die vereinfachte Implementierung von CQRS und dessen Verhalten in Aktion. Zudem sei angemerkt, dass es bessere Kommunikationsmodelle und Datenbankstrukturen für den Event-Bus und die Read-Repositories gibt.

## Dokumentation der Architektur

Im vorliegenden System gibt es ein Frontend, welches als Webseite mit React und Tailwindcss entwickelt wurde. Diese Webseite kommuniziert besonders mit der CommandClientApplication, um Daten zu schreiben und mit den unterschiedlichen QueryClients, um Daten zu lesen.



Die CommandClientApplication schreibt nach jedem Command die gewünschte Änderung in die RelDBApplication. Diese basiert auf der H2-Database (in-memory), welche schnell auch auf PostgreSQL um konfiguriert werden kann. War der Schreibvorgang erfolgreich, so wird das Event über den Event-Bus gepublished und an die gelisteten Subscriber versendet. Bei diesen handelt es sich um die Query Clients, welche sich im Vorfeld subscribed haben. Der Broker im Event-Bus besitzt keinen Broadcast, da dieser nur für das DeleteAll Event hätte benötigt werden können.

Die Webseite kann nun die Daten von den Query Clients abfragen.

## *Kommunikation*

Alle Services und auch die Webseite kommunizieren über REST. Die Daten werden hier meistens über JSON im Body übermittelt und auch vereinzelt über Parameter. Jedes Component, welches Endpoints für die Website zur Verfügung stellt musste entsprechen mit der @CrossOrigin Annotation wegen CORS versehen werden.

Aus dem JSON werden automatisch durch Spring die Klassen serialisiert und deserialisiert.

## *Event-Bus*

Im Event-Bus gibt es den Broker. Dieser speichert die Subscriber mit ihrem Hostname und Endpoint als String in einer Array-Liste zum zugehörigen EventType als Hashmap ab. Dadurch muss durch diese Liste beim publishen nur noch durch geloopt werden.

Möchte man alle Events erneut publishen, so sind diese in einer anderen Liste namens events im EventRepository gespeichert. Diese wird in dem Fall auch durch geloopt.

Der Event-Bus besitzt einen Endpoint, um die Liste der Events direkt an die Website aus Gründen der Einfachheit zu übermitteln

## *QueryClients*

Es gibt drei Query Clients, welche mit der Webseite kommunizieren:

1. QueryClientBookingApplication
2. QueryClientCustomerApplication
3. QueryClientRoomApplication

Alle Clients wurden mit einer selbstimplementierten Linked-List realisiert. Dadurch stehen die Daten auch gleich sortiert zur Verfügung. Bei der QueryClientBookingApplication wurden sogar zwei Verlinkungen verwendet, so dass nach startDate und endDate gesucht werden kann.

## *CommandClientApplication*

In der CommandApplication ist der Initializer erwähnenswert. Dieser resetet und versorgt die Datenbank mit Anfangsdaten. Zudem können nur so die Zimmer gesetzt werden. Der Initializer ist über einen Endpoint zugänglich und wird über den Button Initialize Database von der Webseite ausgelöst.

## *RelDBApplication*

Bei der Datenbank handelt es sich um die in-memory Datenbank H2-Database. Diese ist im Augenblick als in-memory definiert, kann aber auch schnell auf in-file-sytem konfiguriert werden.

Zudem wurde auf eine Model Schicht und ein Repository von Jakarta zurückgegriffen. Dadurch kann diese auch schnell für andere Datenbanken konfiguriert werden. Bei der RelDBApplication handelt es sich auch um einen Service, welcher über Endpoints erreichbar ist.

# Ausführung der Applikation

Zum starten des Backends müssen alle sechs Services gestartet werden. Eine reinfoolge gibt es nicht. Aber man kann dennoch diese Reihenfolge befolgen:

1. RelDBApplication
2. CommandClientApplication

3. EventBus
4. QueryClientBookingApplication
5. QueryClientCustomerApplication
6. QueryClientRoomApplication

Danach kann man auf die Datenbank über die H2-Konsole unter dem folgenden Link zugreifen:

- <http://localhost:8082/h2-console>

Diese ermöglicht eine Visualisierung des Write-Repositories. Das System läuft aber auch ohne öffnen der Konsole.

Nachdem das Backend läuft, kann das Frontend gestartet werden. Dazu muss man sich mit der Powershell oder einer anderen Konsole im Ordner Frontend befinden und erst npm install ausführen. Danach muss man nur noch npm run dev ausführen.

Der Link unter dem die Webseite zu erreichen ist, sollte nun sichtbar sein. Bei uns war es der Link <http://localhost:5173/>.





Sobald die Seite geöffnet wurde, läuft das vollständige System. Alle Button sind Grün markiert und die Eingabefelder grau.

Man sollte immer zu Beginn Initialize Database klicken. Damit auch die Zimmer hinzugefügt werden. Dies ist, aber auch später möglich.

Im Folgenden sind die Abschnitte aufgelistet, welche die jeweiligen gewünschten Aufgaben des Systems vom Aufgabenblatt auflisten:

#### *BookRoom (mit zumindest Zeitraum, Zimmernummer, Kunde)*



##### **Book Room**

Start Date: 15.03.2024  End Date: 31.03.2024  Room Number: 1  Customer: M&M 

Der Customer entspricht dem Username, welcher im System zuvor angelegt wurde. Nur der Username ist einmalig.

#### *CancelBooking (mit Reservierungsnummer)*

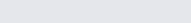
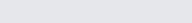


##### **Cancel Booking**

Reservation Number:  

Die Reservierungsnummer kann über Get Bookings eingesehen werden.

#### *CreateCustomer (mit zumindest Name, Adresse, Geburtsdatum)*



##### **Create Customer**

Username:  Name:  Address:  Birthday: TT.MM.JJJJ  

Bitte immer schauen, dass der Username nicht schon vergeben ist.

*GetBookings (Parameter: Zeitraum)*

Get Bookings

Start Date: 07.03.2024  End Date: 31.03.2024 



Load Bookings


Reservation Number	Room Number	Start Date	End Date	Customer
0	2	29.3.2024	31.3.2024	M&M

Die Buchung, welche durch den Button Initialize Database erzeugt wird, startet immer am heutigen Tag und dauert zwei Tage.

*GetFreeRooms (Parameter: Zeitraum, Anzahl Personen)*

Get Free Rooms

Start Date: 06.03.2024  End Date: 11.03.2024 

Persons: 1 

Load Free Rooms

Room Number	Number of Beds	Bath
1	2	yes
2	2	yes
3	2	no
4	2	no
5	4	yes
6	4	yes
7	4	yes
8	4	yes
9	4	yes
10	4	no

Es werden alle Zimmer aufgelistet, welche Platz für die gewünschte Anzahl an Personen haben.

*GetCustomers (Optional Parameter: Name)*

Get Customers

Name:

Load Customers

Username	Name	Address	Birthday
M&M	Max Mustermann	Musterstraße	29.3.2024

## Löschen der Query-Modelle Wiederherstellung durch die im Event-Store gespeicherten Events

Reload Query-Models

### Auslesen aller Events

Reload Events On Frontend

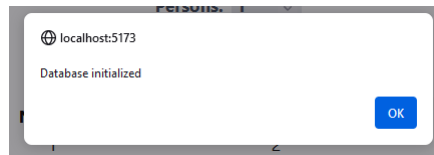
```
{ "eventType": "DELETE_ALL_EVENT", "timestamp": 1711746409955, "eventID": 0 }
{ "eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 0, "number": 1, "numberOfBeds": 2, "bath": true }
{ "eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 1, "number": 2, "numberOfBeds": 2, "bath": true }
{ "eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 2, "number": 3, "numberOfBeds": 2, "bath": false }
{ "eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 3, "number": 4, "numberOfBeds": 2, "bath": false }
{ "eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 4, "number": 5, "numberOfBeds": 4, "bath": true }
{ "eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 5, "number": 6, "numberOfBeds": 4, "bath": true }
{ "eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 6, "number": 7, "numberOfBeds": 4, "bath": true }
{ "eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 7, "number": 8, "numberOfBeds": 4, "bath": true }
{ "eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 8, "number": 9, "numberOfBeds": 4, "bath": true }
{ "eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 9, "number": 10, "numberOfBeds": 4, "bath": false }
{ "eventType": "CREATE_CUSTOMER_EVENT", "timestamp": 1711746409955, "eventID": 0, "username": "M&M", "name": "Max Mustermann", "address": "Musterstraße", "birthday": 1711746409955 }
{ "eventType": "BOOK_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 0, "startDate": 1711746409955, "endDate": 1711919209955, "roomNumber": 2, "customer": "M&M", "reservationNumber": 0 }
{ "eventType": "CREATE_CUSTOMER_EVENT", "timestamp": 1711746438098, "eventID": 1, "username": "HP", "name": "Harry Potter", "address": "Little Whinging", "birthday": 3338496000000 }
{ "eventType": "BOOK_ROOM_EVENT", "timestamp": 1711746466653, "eventID": 1, "startDate": 1710288000000, "endDate": 1710374400000, "roomNumber": 1, "customer": "HP", "reservationNumber": 1 }
```

## Erzeugung der Command- und Query-Modelle (anhand hinterlegter Basisdatensätze)

Initialize Database

## Verhalten der Applikation anhand eines Scenarios

Das Hotel „SoUndSo“ hat das Reservation System erfolgreich gestartet. Die vordefinierten Zimmer entsprechen auch den Zimmern vom Hotel und Max Mustermann existiert mit seiner Buchung tatsächlich. Also klickt der Hotelier auf Initialize Database und bekommt die Rückmeldung einer erfolgreichen Initialisierung über einen Alert.



Danach überprüft dieser, ob die Zimmer auch wirklich existieren und wählt eine Zeitspanne vor dem hütigen Datum mit einer Person aus.

Get Free Rooms

Start Date: 13. 03. 2024 End Date: 28. 03. 2024

Persons: 1

Load Free Rooms

Room Number	Number of Beds	Bath
1	2	yes
2	2	yes
3	2	no
4	2	no
5	4	yes
6	4	yes
7	4	yes
8	4	yes
9	4	yes
10	4	no

Nun kommt der Gast Harry Potter und möchte für den 13.03.2024 für eine Person reservieren. Der Hotelier weis schon, dass das Zimmer eins frei ist und legt den Gast Harry Potter an.

#### Create Customer

Username:  Name:  Address:  Birthday:

Danach überprüft dieser, ob die Daten des Kunden abgespeichert wurden. Zu seinem Glück existiert nur ein Harry Potter und der Hotelier muss nicht scrollen.

Get Customers

Username	Name	Address	Birthday
HP	Harry Potter	Little Whinging	31.7.1980

Nun wird reserviert.

#### Book Room

Start Date:  End Date:  Room Number:  Customer:

Und anschließend sieht der Hotelier die existieren Buchungen für den 7.3.2024 bis 31.3.2024 ein.

Get Bookings

Reservation Number	Room Number	Start Date	End Date	Customer
1	1	13.3.2024	14.3.2024	HP
0	2	29.3.2024	31.3.2024	M&M

Der Prozessmanager will nun Abläufe optimieren und bieten den Hotelier ihm seinen Tagesablauf zu zuschicken. Dieser schickt stattdessen einfach eine Kopie der Event Liste im System

#### Special Functions

```
{
  "eventType": "DELETE_ALL_EVENT",
  "timestamp": 1711746409955,
  "eventID": 0
}
{"eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 0, "number": 1, "numberOfBeds": 2, "bath": true}
{"eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 1, "number": 2, "numberOfBeds": 2, "bath": true}
{"eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 2, "number": 3, "numberOfBeds": 2, "bath": false}
{"eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 3, "number": 4, "numberOfBeds": 2, "bath": false}
{"eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 4, "number": 5, "numberOfBeds": 4, "bath": true}
{"eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 5, "number": 6, "numberOfBeds": 4, "bath": true}
{"eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 6, "number": 7, "numberOfBeds": 4, "bath": true}
{"eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 7, "number": 8, "numberOfBeds": 4, "bath": true}
{"eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 8, "number": 9, "numberOfBeds": 4, "bath": true}
{"eventType": "ADD_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 9, "number": 10, "numberOfBeds": 4, "bath": false}
{"eventType": "CREATE_CUSTOMER_EVENT", "timestamp": 1711746409955, "eventID": 0, "username": "M&M", "name": "Max Mustermann", "address": "Musterstraße", "birthday": 1711746409955}
{"eventType": "BOOK_ROOM_EVENT", "timestamp": 1711746409955, "eventID": 0, "startDate": 1711746409955, "endDate": 1711919209955, "roomNumber": 2, "customer": "M&M", "reservationNumber": 0}
{"eventType": "CREATE_CUSTOMER_EVENT", "timestamp": 1711746438098, "eventID": 1, "username": "HP", "name": "Harry Potter", "address": "Little Whinging", "birthday": 333849600000}
{"eventType": "BOOK_ROOM_EVENT", "timestamp": 1711746466653, "eventID": 1, "startDate": 1710288000000, "endDate": 1710374400000, "roomNumber": 1, "customer": "HP", "reservationNumber": 1}
```