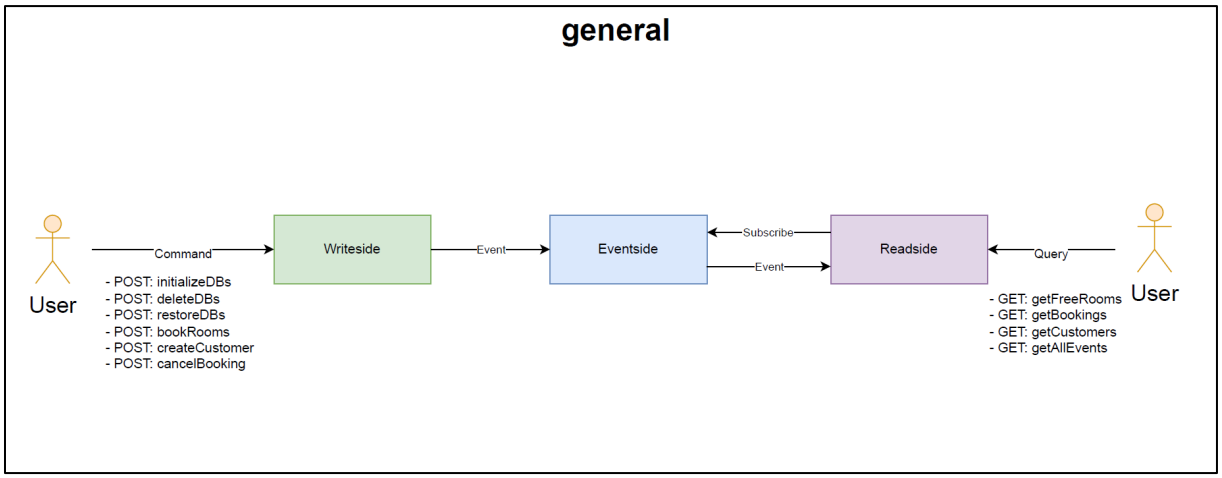
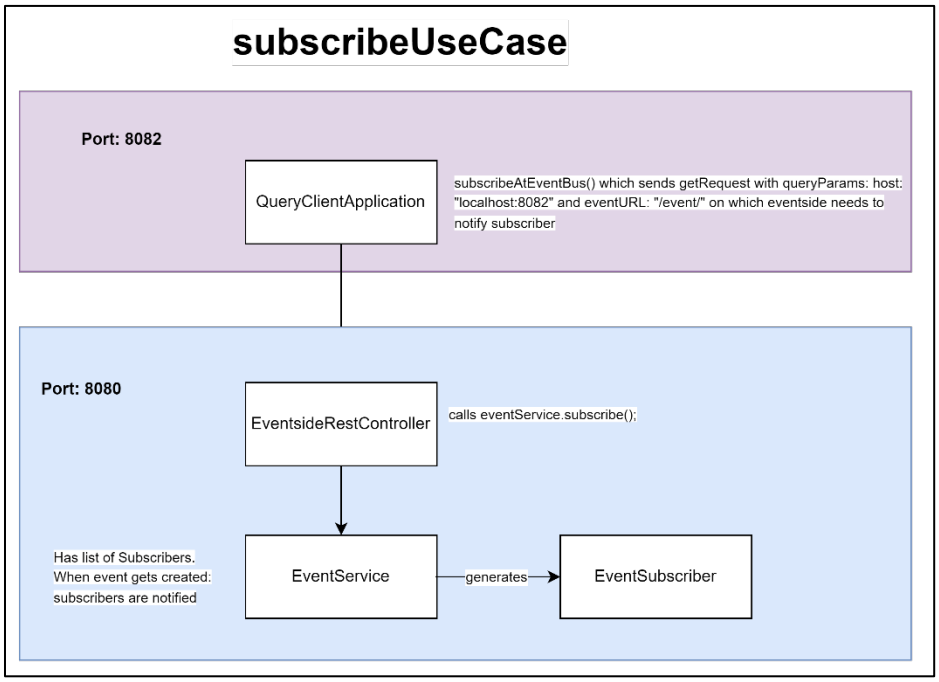
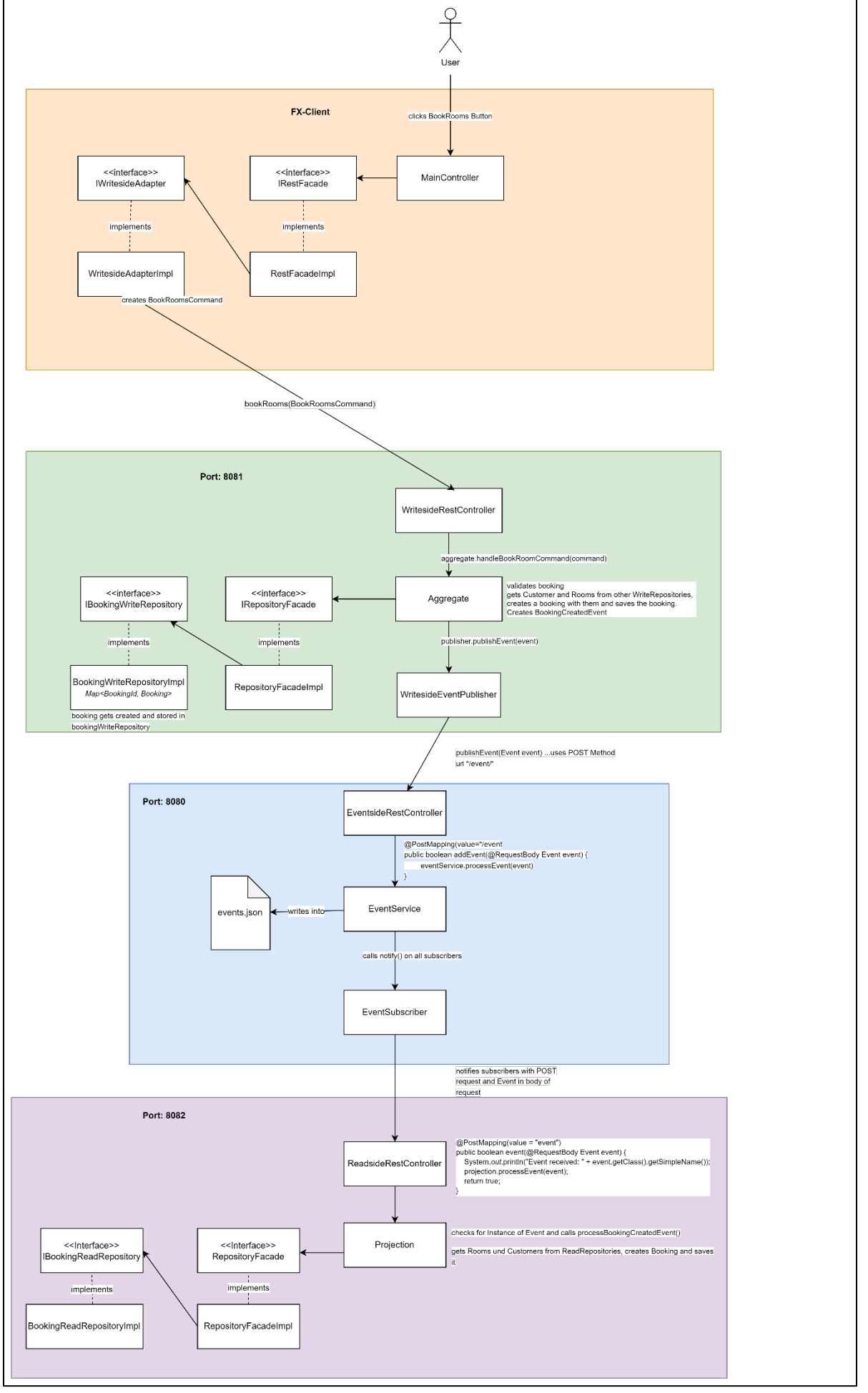
Dokumentation CQRS and Event Sourcing

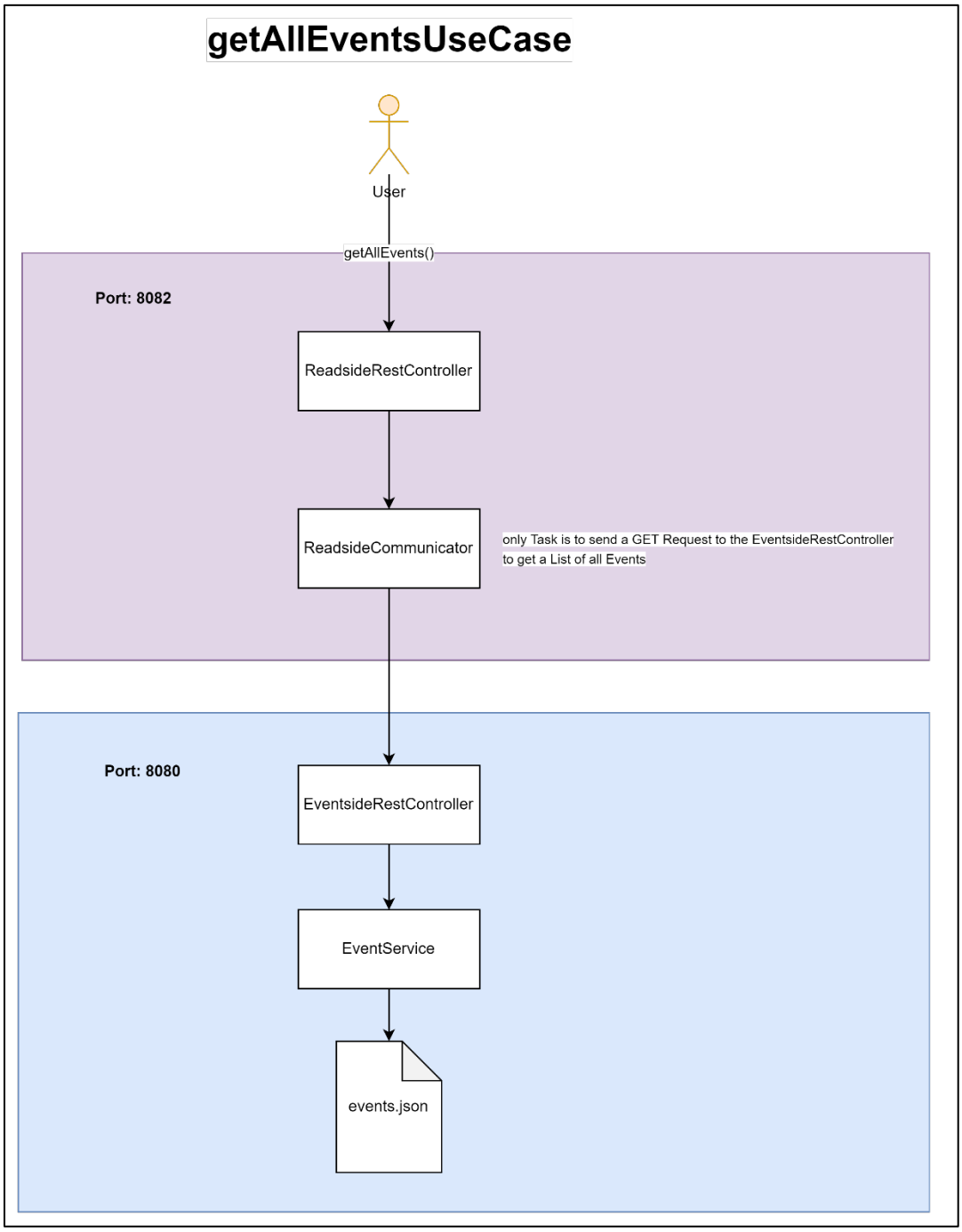
Clara Tschamon





bookRoomsUseCase





Beschreibung zur Ausführung der Applikation:

Den EventBus, die QueryClientApplication und die CommandClientApplication starten.

Wichtig: Die QueryClientApplication darf nicht vor dem EventBus gestartet werden, da sich sie sich bei Start auf den EventBus subscribed.

Als letztes wird die Main Application im FXClient gestartet.

Es öffnet sich automatisch das User Interface.

Zu Beginn sind noch alle Repositories leer.

- ➔ Initialize DBs Button drücken
- ➔ InitializeDBs wird auf der „writeside“ also auf dem QueryClient aufgerufen. Auf der writeside werden die Listen in den repositories mit den in den repositories statisch festgelegten Daten initialisiert.

Für jedes initialisierte Objekt wird ein Event erzeugt, welches vom „WritesideEventPublisher“ an die „eventside“ sendet. Dort schreibt der „EventService“ das Event in die events.json File und sendet es weiter an alle Subscriber (an die „readside“)

➔ Auf der „readside“ wird das Event von der „Projection“ bearbeitet.

Info:
Der Inhalt der events.json File wird bei jedem start des Eventbus gelöscht.

Beschreiben der Applikation an einem einfachen Szenario:
Szenarios „Räume buchen“ „subscriben“ und „alle Events anzeigen“ sind schon in den Grafiken oben dargestellt.
Szenario: Freie Räume in einem Zeitraum abfragen

- ➔ User gibt Anreisedatum, Abreisedatum und die Anzahl der Personen in Userinterface ein. Auf Buttonklick ruft der UseCaseController Methode in RestFacade auf. Von da an Wird der Request weitergeleitet an den ReadsideAdapter.
- ➔ Der ReadsideAdapter sendet GET Request an den ReadsideRestController. Die Parameter werden als Requestparameter in der URL übertragen.
- ➔ Der ReadsideRestController erstellt GetFreeRoomsQuery und übergibt diese an die Projection Klasse.
- ➔ Die Projection Klasse ruft getFreeRooms() auf der RepositoryFacade auf, wodurch getFreeRooms() im BookedRoomsReadRepository aufgerufen wird.
- ➔ Im BookedRoomsReadRepository gibt es eine HashMap mit Datum als Key und Liste von gebuchten Raumnummern als Value. Um die freien Räume rauszufiltern, wird aus dem RoomReadRepository eine Liste mit allen Räumen geholt, eine Kopie der Liste erstellt und für jeden Tag im angefragten Zeitraum überprüft, ob der Raum bereits gebucht ist. Wenn er schon gebucht ist, wird er aus der Liste rausgelöscht. So bleiben die freien Räume übrig.
- ➔ Die Liste aus freien Räumen wird returned.

Info: Damit besser statische bookings eingespeichert werden konnten, ist die BookingId anstatt einer UUID ein String. Die UUID ist zu lang, um sie zum cancellen eingeben. BookingIds werden aber mit „UUID.randomUUID().toString()“ generiert.

Erklärung:
Warum habe ich für Repositories Sets anstatt Lists verwendet?

➔ damit z.B. der gleiche room nicht ausversehen mehrmals eingespeichert werden kann.
Für repository Funktionen ist es besser, Sets zu verwenden, um Duplikationen zu vermeiden.

Erklärung zur @JsonTypeInfo Annotation:
Damit EventsideRestController das Event nach dem empfangen zurück auf die konkrete Klasse mappen kann. Ohne dieser Annotation würde die spezifische subklasse von Event Klasse nicht erkannt werden.

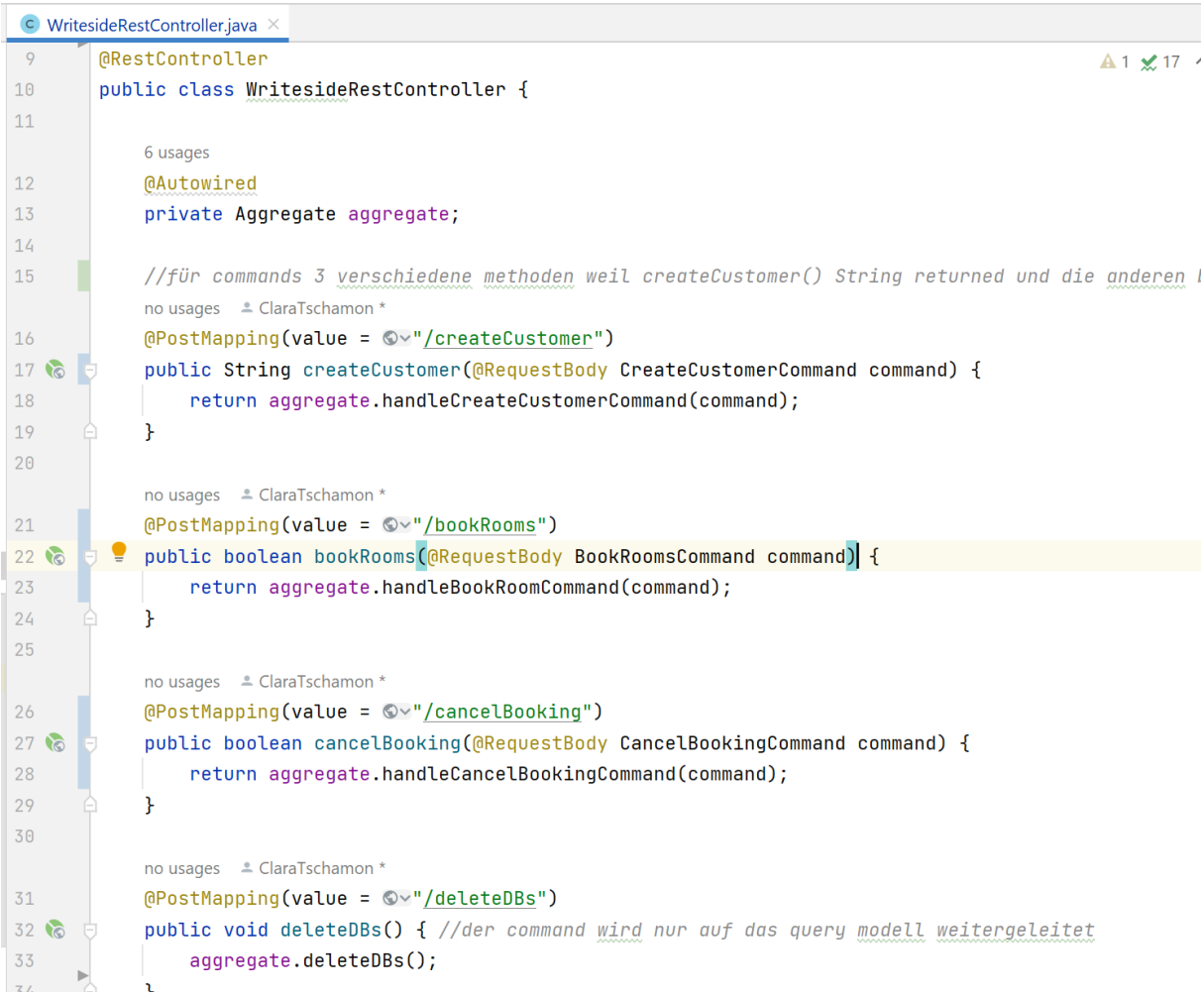
Event.java

```java
package at.fhv.cts.eventside.events;

import com.fasterxml.jackson.annotation.JsonTypeInfo;

import java.time.LocalDateTime;

6 inheritors  ClaraTschamon
@JsonTypeInfo(use= JsonTypeInfo.Id.CLASS, property="className")
public abstract class Event {

    4 usages
    private LocalDateTime timestamp;

    ClaraTschamon
    protected Event(LocalDateTime timestamp) { this.timestamp = timestamp; }
```

EventsideRestController.java EventService.java

```java
2 usages  ClaraTschamon
@Service
public class EventService {

    6 usages
    private Map<String, EventSubscriber> subscribers = new HashMap<>();


    1 usage  ClaraTschamon
    public void processEvent(Event event) {

        if (!(event instanceof DBsDeletedEvent)) {
            writeEvent(event);

        }
        for (EventSubscriber subscriber : subscribers.values()) {
            subscriber.notify(event);
        }

        if (event instanceof DBsRestoredEvent) {
            restoreEvents();
        }
    }
```

Erklärung: Der Einzige Grund, warum nicht nur eine Methode value="/command" mit Eingabewert: abstract class Command verwendet wurde, ist, dass createCustomer() einen anderen Rückgabewert hat als bookRooms() und cancelBooking().



Nice to have:

➔ customer speichern erst wenn bookRoom erfolgreich validiert wurde... so erstellt es mir jetzt immer customer obwohl room noch nicht gebucht ist -> jetzt lösche ich den customer in der validierung wieder wenn buchung nicht erstellt werden kann

Quellen:

https://www.baeldung.com/spring-component-repository-service
https://www.baeldung.com/cqrs-event-sourcing-java
https://howtodoinjava.com/spring5/webmvc/controller-getmapping-postmapping/
https://www.baeldung.com/spring-request-param
https://howtodoinjava.com/jackson/java-8-date-time-type-not-supported-by-default/