Lab 01 - CQRS und Event-Sourcing

Einleitung

Dieses Seminar beschäftigt sich mit den Architektur-Konzepten *CQRS (Command Query Responsibility Segregation)* und *Event-Sourcing*. Der Grundgedanke von CQRS besteht darin, Anwendungen und die dazugehörigen (Domain-)Modelle entsprechend aufzuteilen. Dabei werden das Datemodell verändernde Kommandos (*Commands*) von reinen Abfragen (*Queries*) getrennt und jeweils dezidierte Modelle für diese erstellt. Diese Modelle sind darauf ausgelegt möglichst effizient gewisse Funktionalitäten abzudecken. Beispielsweise müssen Commands vor deren Ausführung auf einem Modell validiert werden. Nur wenn diese Validierung erfolgreich ist, dürfen die Kommandos das dahinterliegende Datenmodell aktualisieren. Hierfür werden häufig klassiche Domänenmodelle oder relationale Datenbankrepräsentation verwendet. Für Abfragen ist hingegen häufig die Laufzeit entscheidend. Um diese Abfragen möglichst effizient durchführen zu können, werden Datenmodelle erstellt die speziell für vorgegebene Abfragen ausgelegt sind. Dadurch ist es möglich die Datenrepräsentation so zu wählen, dass beispielsweise wenige Datenquellen (Tabellen) verknüpft werden müssen. Die Synchronisierung der Modelle für die Kommandos und Abfragen wird dabei über *Events* realisiert. Für Queries werden die entsprechenden Events über eine eigene *Projection* auf dezidierte Datenmodelle abgebildet. Es kann dabei für jeden Query ein eigenes Datenmodel notwendig werden.

Grundsätzlich kann die Aufteilung auf Commands und Queries innerhalb einer Applikation erfolgen. In diesem Seminar werden wir allerdings getrennte, über *REST* kommunizierende, Applikationen für Commands und Queries erstellen. Die Interaktion mit dem System erfolgt über einen (*Web-)Client*. Das System ist in Abbildung 1 dargestellt.

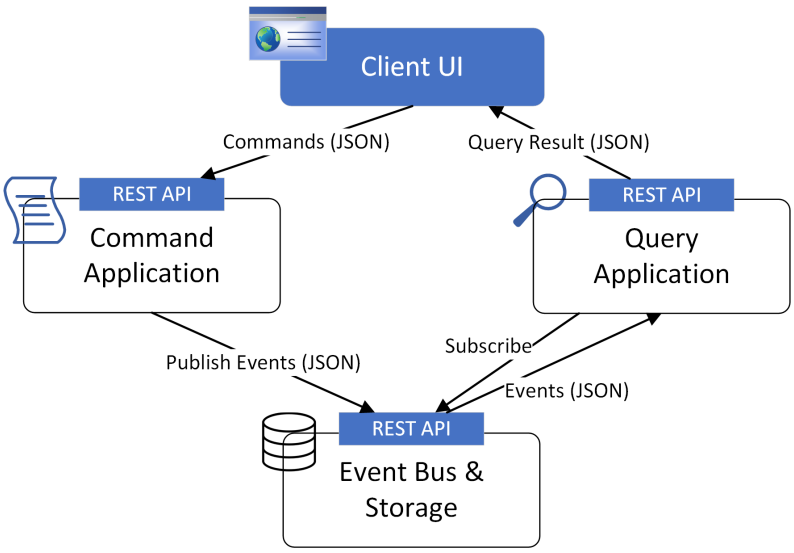

Abbildung 1: Aufteilung auf einzelne Applikationen unter Verwendung von CQRS

Event-Sourcing wird häufig in Kombination mit CQRS verwendet. Grundgedanke dabei ist, dass der Zustand eines Systems nicht wie überlicherweise durch relationale Domänenmodelle synchronisiert wird, sondern das Änderungen (*Events*) die auf den Domänenmodellen ausgeführt

werden gespeichert werden. Der aktuelle Zustand des Systems muss also nicht explizit in einer relationalen Datenbank persistiert, sondern kann auch nur *in-memory* existieren. Durch die Speicherung aller Änderungen (*Events*) kann zu jedem Zeitpunkt der aktuelle Zustand eines Systems durch die sequentielle Ausführung dieser Events wiederhergestellt werden. Dadurch bleibt jede Änderung am Zustand nachvollziehbar. Wird beispielsweise in einem klassischen Modell der Name einer Person geändert, ist nach der Änderung nur noch der neue Name ersichtlich. Bei Event-Sourcing hingegen, wird nicht der neue Name, sondern ein Event welches diese Änderung beschreibt gespeichert und dadurch bleibt die Änderung nachvollziehbar. Abbildung 2 gibt einen Überblick über dieses Architekturmuster.

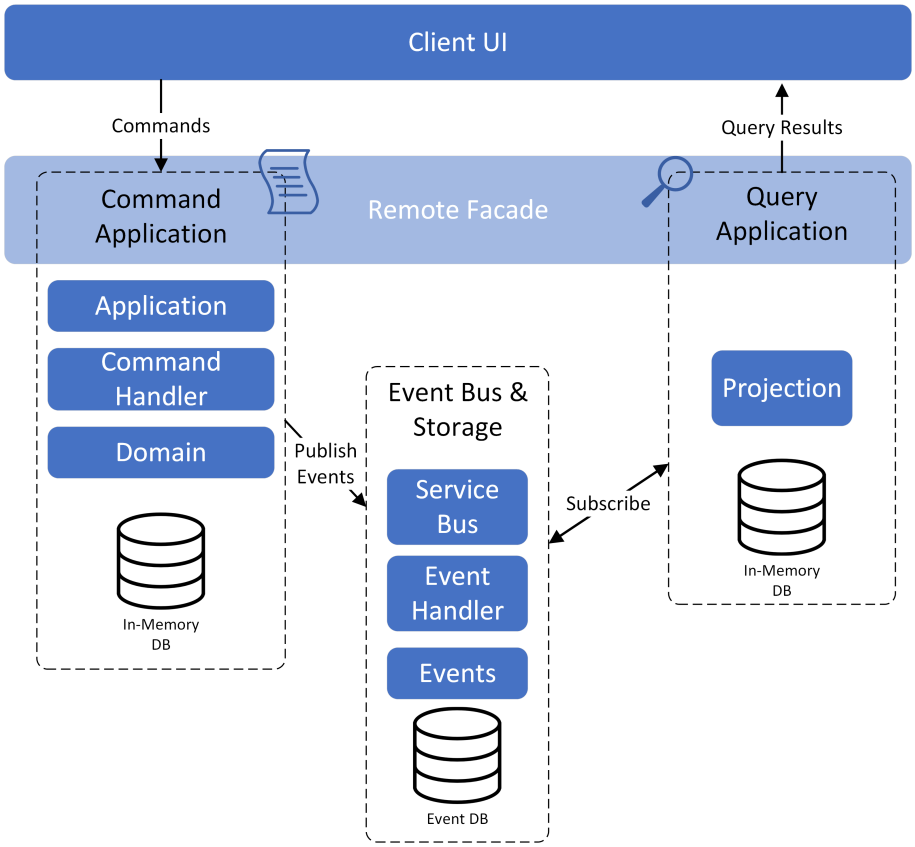

Abbildung 2: CQRS in Kombination mit Event-Sourcing

In diesem Seminar verwenden wir eine leicht abgewandelte Form von Event-Sourcing. Wir gehen für die Command-Application davon aus, dass der Zustand des Systems immer in-memory vorhanden und aktuell ist. Wir müssen also nicht für jedes neue Kommando den internen Zustand immer wieder aus den vorangegangenen Events aufbauen oder eine relationale Datenbank verwenden.

Aufgabe

Start: 2025-03-14

Ende: 2025-04-24

Implementieren Sie ein Reservierungssystem für Hotelzimmer mittles CQRS und Event-Sourcing. Die Interaktion mit dem System soll über ein UI (Kommandozeile, Web-Interface, Desktop-GUI) möglich sein. Teilen Sie Ihr System auf mindestens drei Applikationen (Commands, Queries, Events) auf, die mittels REST miteinander kommunizieren.

Beschreibung

Das Reservierungssystem für Hotelzimmer ist einfach gehalten und baut auf einem einfachen Domänenmodell auf. Es soll möglich sein Zimmer für einen bestimmten Zeitraum zu buchen. Jedes Zimmer hat dabei eine Zimmernummer, einen Preis und mindestens noch drei weitere Eigenschaften (z.B. Anzahl an Personen). Jede Buchung erfolgt für einen Kunden der ebenfalls im System verwaltet werden kann. Die Verwaltung der Räume erfolgt ebenfalls über das System. Die Daten müssen dabei nicht dauerhaft persistiert werden, das Speichern des aktuellen Zustands für die Laufzeit des Programmes ist ausreichend.

Das System soll mithilfe des CQRS-Patterns umgesetzt werden. Dabei sind mindestens folgende Commands zu unterstützen:

```
BookRoom (mit zumindest Zeitraum, Zimmernummer, Kunde)
CancelBooking (mit Reservierungsnummer)
CreateCustomer (mit zumindest Name, Adresse, Geburtsdatum)
UpdateCustomer
PayBooking (mit zumindest Zeitpunkt, Bezahlungsart...)
```

Die Commands sollen auf einem einfach Domänenmodell, bestehend aus zumindest Zimmern, Buchungen, Kunden und Bezahlung inkl. Rechnung, validiert werden. Kann ein Command erfolgreich ausgeführt werden, wird ein Event erzeugt. Dieses Event wird für den Aufbau der Modelle für die Queries verwendet.

Um mit dem System interagieren zu können, muss es möglich sein Daten von diesem abzufragen. Diese Abfragen werden über Queries ermöglicht. Es müssen dabei mindestens folgende Queries unterstützt werden:

```
GetBookings (Parameter: Zeitraum):
  Zeigt alle Buchungen im gewählten Zeitraum an
GetFreeRooms (Parameter: Zeitraum, Anzahl Personen):
  Zeigt die verfügbaren Zimmer für die angefragten Daten an
GetCustomers (Optional Parameter: Name):
  Liefert alle im System gespeicherten Kunden zurück
```

Es wird notwendig sein, verschiedene Read-Modelle für die unterstützten Queries aufzubauen. Die Daten müssen dabei nicht in eine relationalen Struktur abgebildet werden, sondern können "flach", als eine einzige Entität, dargestellt werden. Die Struktur der Daten soll dabei so gewählt werden, dass die Abfragen möglichst unkompliziert und effizient durchgeführt werden können. Die Synchronisierung dieser Modelle erfolgt über eine eigene Projection, die die von der Command-Applikation erstellten Events verarbeitet. Diese Datenmodelle für die Queries werden in einer relationalen Datenbank persistiert.

Die Events werden über eine eigene Applikation, ein sehr einfacher *Service-Bus*, verteilt. Dabei soll es möglich sein sich für bestimmte Event-Typen zu subscriben und bei einem entsprechenden Event benachrichtigt zu werden. Der Service-Bus ist als eigenständige Applikation zu implementieren und verarbeitet alle von der Command-Applikation erzeugten Events über einen Event-Handler. Die Events müssen dabei in geeigneter Form persistiert werden (in einem dezidierten Event-Store z.B. DB). Die gespeicherten Events sollen geladen und zur Wiederherstellung von z.B. Query-Modellen verwendet werden können. Ein anderer Anwendungsfall wäre wenn sich eine zusätzliche Query-Applikation, einige Zeit nachdem bereits

Commands ausgeführt wurden, verbindet und das interne Query-Modell aufgebaut werden muss.

Die Stammdaten, also die verfügbaren Zimmer, bestehende Buchungen und Kunden, können statisch im System hinterlegt werden und durch einen Trigger über das Interface erzeugt werden. Es müssen in diesem Fall aber immer Events erzeugt werden, da ansonsten der Zustand nicht nachvollziehbar ist. Bieten Sie über das Interface folgende Funktionalität an:

```
Löschen der Command und Query-Modelle
Wiederherstellung durch die im Event-Store gespeicherten Events
Auslesen aller Events
Erzeugung der Command- und Query-Modelle (anhand hinterlegter Basisdatensätze)
```

Bei einem Neustart der Applikation kann der Speicher überschrieben und neu initialisiert werden.

Für die Implementierung dieser Aufgabenstellung ist in Ilias ein Template verfügbar, welches drei getrennte Applikationen startet und zeigt wie diese über REST kommunizieren können. Verwenden Sie dieses um Ihr System zu entwickeln und erweitern/adaptieren Sie es entsprechend. Für die Implementierung des Frontends kann eine beliebige Technologie gewählt werden. Die Wahl der verwendeten DB kann frei erfolgen.

Abgabe

Die Abgabe erfolgt in Ilias bis 2025-04-24. Geben Sie den Source-Code als gradle Projekt, eine Dokumentation der Architektur und eine Beschreibung zur Ausführung der Applikation ab. Beschreiben Sie das Verhalten der Applikation an einem einfachen Szenario.