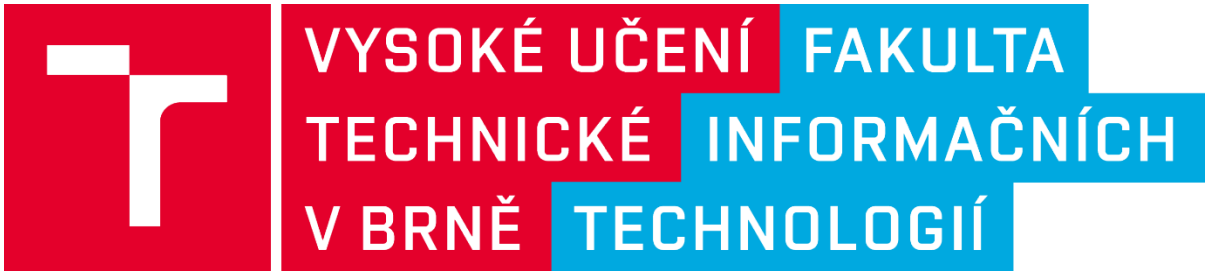


Projekt PDB

Návrh

Správa pokojů v hotelu

Autor: Bc. Martin Bobčík, xbobci00

23.10.2020

Specifikace

Téma aplikace je správa hotelů a pokojů. Aplikace obsahuje hotely. V každém hotelu je několik pater a pokojů. Pokoje mohou být buď obsazené hosty, nebo prázdné. Každý pokoj má svou cenu a může být buď uklizený, nebo neuklizený.

Se systémem se komunikuje pomocí zpráv typu Command a Query. Commands mění stav systému, a míří do SourceOfTruth, SQL databáze. Query se pouze dotazují na stav systému a nemění jej. Tyto zprávy míří na Reading Cache, NoSql databázi Cassandra.

Dostupné příkazy

- [Put]Api/Command/room/
    - Occupy/id – ubytuje zákazníka na pokoj s id. Id zákazníka je posláno jako Json body requestu jako atribut guest
    - Free/id – od ubytuje zákazníka z pokoje id a označí jej pro úklid
    - Clean/id – označí pokoj id jako čistý
    - Dirty/id – označí pokoj id jako špinavý
    - Price/id – změní cenu pokoje id. Cena je poslána v json body jako atribut price
- Api/Command/hotel
    - [Post]Room – vytvoří nový pokoj v hotelu. Vlastnosti pokoje jsou v json body požadavku
    - [Delete]Room/id – smaže pokoj s Id z databáze
    - [Post] – přidá do systému nový hotel, jehož atributy jsou v těle požadavku
    - [Delete]/id – smaže ze systému hotel s id
- [Get] Api/query/room– vrátí seznam pokojů v hotelu. Jeho Id nebo jméno se specifikuje pomocí json atributů id/name v těle požadavku
    - Unoccupied – vrátí seznam volných, čistých pokojů, a to buď z celého systému, nebo jen jednoho hotelu specifikovaného pomocí id/name v těle požadavku
    - Unoccupied/cheapest – Stejné jako Unoccupied, ale vrátí pouze nejlevnější pokoj
    - Toclean – vrátí seznam pokojů určených k úklidu. Opět lze specifikovat hotel.

Scénář předpokládaného použití

Systém obsahuje množství různě obsazených pokojů. Zákazník přijde na recepcih otelu a chce se ubytovat. Recepce se dotáže na volné pokoje (RoomQuery.GetListUnoccupied). Zákazník si přeje nejlevnější (RoomQuery.GetUnoccupiedCheapest). Recepce nově nalezený pokoj přidělí zákazníkovi (RoomCommand.Occupy).

Zákazník si přeje uklidit svůj pokoj. Na (například) informačním panelu pokoje zadá požadavek (RoomCommand.Dirty). Uklízečka průběžně kontroluje stav úklidu (RoomQuery.GetListDirty), zjistí, že je potřeba jejích služeb a vydá se konat. Po dokončení oznámí, že je pokoj čistý (RoomCommand.Clean).

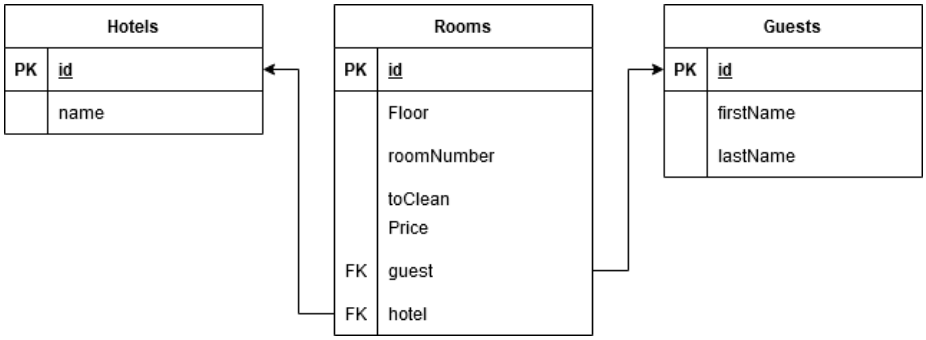
Zákazník se rozhodne svůj pobyt ukončit. Přijde na recepci a představí svůj požadavek (RoomCommand.Free). Tím se pokoj uvolní, ale také se pokoj označí k úklidu, a znova proběhne kolo s uklízečkou. Po uklizení se pokoj opět objeví v RoomQuery.GetListUnoccupied. Mimo to se také o tento pokoj zvýšil zájem a zvýší se jeho cena (RoomCommand.ChangePrice). Po této změně se už neukáže v RoomQuery.GetUnoccupiedCheapest.

Tento scénář, ve stejném pořadí a s popisem, lze najít ve složce PostmanCollections s názvem CommonUsage. Druhý scénář ve složce realizuje vytvoření a zrušení hotelu a pokoje. Scénáře je

možné spustit pomocí aplikace Postman. Při testování je potřeba dávat pozor na přijímané a odesílané id jednotlivých entit.

Analýza

Relační databáze obsahuje jednu hlavní tabulku Rooms, která drží veškeré pokoje systému. Každý pokoj je popsán jeho unikátním identifikátorem, patrem, číslem pokoje, cenou, značkou úklidu. Dále má dva cizí klíče odkazující na hotel, ve kterém se pokoj nachází a na hosta, který je v pokoji ubytován. Dále existuje tabulka hotels, popisující jednotlivé hotely v systému a tabulka guests popisující klienty.



NoSQL databáze ukládá data k jednotlivým dotazům (podle query-first přístupu). Databáze obsauje několik tabulek:

- Rooms.byHotel – ukládá pokoje do skupin podle hotelů v nichž se nachází. Každý řádek také obsahuje ubytovaného hosta.
- Rooms.toOcuppy – obsahuje seznam pokojů, na které se lze ubytovat. Tyto jsou opět seřazené podle hotelů. Tato tabulka pochopitelně neobsahuje informace o hostech.
- Rooms.toClean – seznam pokojů, které je potřeba vyčistit. Stejný princip jako předchozí tabulka. Tato tabulka neobsahuje data o ceně pokoje.
- Rooms.byGuest – ukládá pokoje seřazené podle hostů (tato nakonec zůstala nevyužita)

Na tyto tabulky se pak aplikace dotazuje jednoduchými selecty případně omezenými na hotel. Data v nerelační databázi vznikají na základě požadavků Commands, tedy ne rychleji než rychlost těchto požadavků. Data jsou platná, dokud nepřijde požadavek je změnit. Relační data se aktualizují pomocí posílání Commands požadavků na příslušné API. To, po vyřízení požadavku v databázi, kontaktuje jiné API, které provede projekci dat do nerelační databáze.

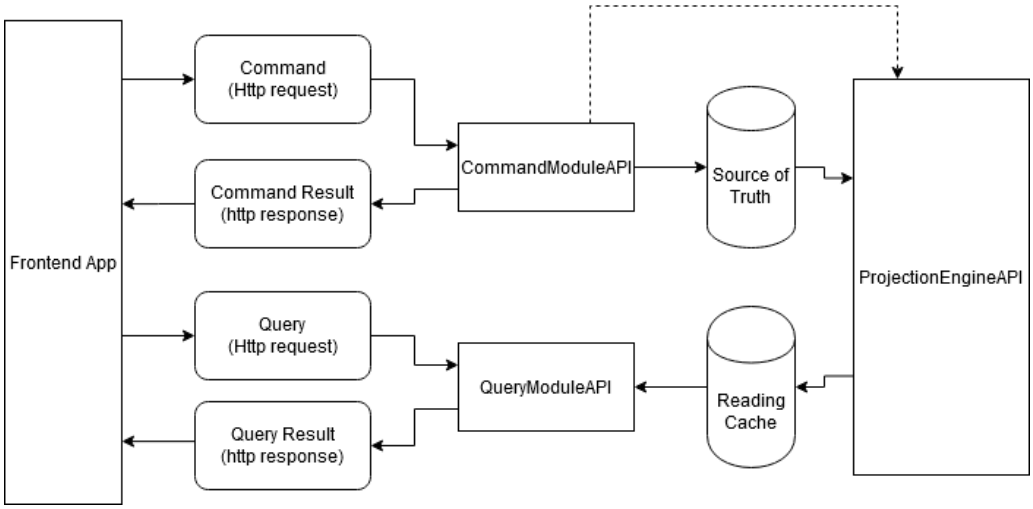
Návrh

Jako nerelační databáze je použita Cassandra, která podporuje shlukování dat podle Partition key, což je v tomto případě hotel, ve kterém se pokoj nachází. Hotelem je myšleno jeho jméno, a po zaindexování hotel.id také jeho id. Uvnitř Partition(hotelu) jsou pokoje seřazeny pomocí Clustering Key (patro, číslo pokoje). Všechny vyjmenované tabulky budou v jednom prostoru klíčů. Relační databázi zajišťuje MySQL.

Celý systém tvoří, kromě NoSQL a SQL databází, tři aplikační rozhraní. První obsluhuje pouze Query požadavky. Dotazuje NoSQL databázi, provádí dodatečné výpočty nad daty a odesílá výsledek zpět. Druhé API obsluhuje všechny Command požadavky. Zpracuje data tak, aby je mohl vložit do databáze a uloží je. Po vyřízení každého Command požadavku je však nerelační databáze

v nekonzistentním stavu vůči relační, Source of Truth. Odešle se tedy požadavek k projekci dat. O to se stará poslední API. Po přijetí požadavku vyžádá si data v požadovaném tvaru z relační databáze, a uloží je do tabulek nerelační databáze. Jednotlivé tabulky NoSQL neobsahují všechny řádky nebo sloupce ze SQL.

Tímto je dosaženo segregace odpovědností podle návrhového vzoru CQRS.



Každý požadavek typu Command vrací buď pravdivostní hodnotu na základě úspěšnosti požadavku, nebo identifikační číslo vytvořené entity.

V databázi existuje speciální klient s id=1, který značí neobsazené pokoje. Lze tedy obsadit pouze ty pokoje, které mají toto id hosta (řešeno pomocí SQL podmínky). Pokud tedy přijdou dva požadavky na obsazení stejného pokoje, pomalejší z nich nemůže změnit databázi, a vrátí se negativní.

Systém je testován pomocí volání http požadavků pomocí programu Postman[1], jednotlivé scénáře testů představují kolekce požadavků na implementovaná API. Na sofistikovanější řešení testů bohužel nezbyl čas.

Celý systém byl implementován jazykem C#.Net na operačním systému Windows 10. Cílový framework aplikačních rozhraní je .Net Core 2.1 a jsou vytvořeny pomocí knihoven Microsoft.AspNetCore. Pro přístup do nerelační databáze byla použita knihovna CassandraCSharpDriver by DataStax[2]. Pro relační databázi byla použita knihovna MySql.Data by Oracle[3]

Postup spuštění systému

Relační databáze

1. Pomocí XAMPP spustíme službu Apache a MySQL na adrese localhost a portu 3306.
2. Vytvoříme databázi se jménem hotelis.
3. Existuje uživatel root s prázdným heslem a právy k právě vytvořené databázi.
4. Provedeme všechny příkazy v souboru hotelis_more data.sql

[1] https://www.postman.com/downloads/
[2] https://github.com/datastax/csharp-driver
[3] https://dev.mysql.com/downloads/

Nerelační databáze

1. Spustíme Cassandru na adrese 127.0.0.1
    a. Spustíme systém podle návodu na https://rychly-edu.gitlab.io/dbs/nosql/nixos-dbs-vm/
    b. Nastavíme předávání portů 9042⇔9042
2. Spustíme cqlsh
3. Postupně provedeme příkazy v souboru init.cql

Stažení balíků

1. Otevřeme projekt ve Visual Studio
2. V okně Solution Explorer najdeme Solution `HotelIS`
3. Klikneme pravým
4. Restore NuGet Packages

Snažil jsem se dojít na to, jak zautomatizovat tuto část, ale příkaz dotnet restore nikdy nestáhnul všechny potřebné balíky a následný překlad díky tomu havaroval.

Překlad a spuštění

- Když už je otevřené Visual Studio zmáčkneme klávesu F6 (výsledné spustitelné soubory a knihovny budou ve složkách zdrojových kódů HotelIS\(projekt)\bin\Debug
- Také jde spustit původně zamýšlený soubor publish.bat. Ten uloží jednotlivé API do složky bin jako spustitelné exe soubory. Ty lze jednoduše bez hledání spustit soubory start*API.bat

Po spuštění API poslouchají na:

- CommandModuleAPI: https://localhost:5001;http://localhost:5000
- QueryModuleAPI: http://localhost:5050
- ProjectionEngineAPI: http://localhost:5005

Známé nedostatky řešení

- Nedostatečný rozsah testů systému
- Řešení testů pomocí Postman kolekcí není hodné školního projektu
- Složité stažení balíků před prvním spuštění
- Při každém volání Command požadavku se projektují veškerá data
- Při projekci dat se smažou data ze všech NoSQL tabulek a opět se naplní
- Pozdní začátek řešení