Předmět: PDB
Autoři: Ondřej Studnička (xstudn00), Jan Lorenc (xloren15)
Datum: 4. 12. 2022

Hotel - návrh databázového systému

Specifikace

Tématem projektu je vytvořit aplikaci, která by umožňovala správu hotelu, tzn. pokojů a rezervací. V hotelu se nachází několik pokojů, ke kterým si zákazníci vytváří rezervace. Vedení/zaměstnanci spravují jednotlivé rezervace. Provádí check-in a check-out zákazníka, přijímají doplatky rezervací, mohou označit pokoj jako uklizený/neuklizený, provádět klasické CRUD operace nad pokoji a podobně.

Případy užití

- vytvoření rezervace
- zaznamenání příjezdu/odjezdu zákazníka
- storno rezervace
- evidování doplatku za pobyt
- označení pokojů jako uklizených/neuklizených
- CRUD operace nad pokoji
- získání všech aktuálně ubytovaných zákazníků
- získání všech aktuálně volných pokojů
- získání volných pokojů na základě parametrů (např. počet postelí)

Scénář nejčastějšího užití

Zákazník se rozhodne ubytovat v hotelu. Zvolí si termín příjezdu a odjezdu a další případné požadavky (např. cena, počet postelí, patro - vhodné pro lidi na vozíčku, …). Na základě zadaných parametrů zobrazí systém zákazníkovi seznam aktuálně dostupných pokojů, které odpovídají jeho požadavkům. Zákazník si vybere jeden z pokojů a pro něj vytvoří rezervaci.

Následně při příjezdu zákazníka provede recepční check-in proceduru. Zkontroluje, zda je pokoj uklizen. Pokud ano, tak recepční potvrdí check-in, čímž změní stav rezervace.

V případě, že zákazník je ubytován dlouhodobě, tak může na recepci požádat o úklid pokoje. Recepční tento požadavek zaznamená. Uklízečka pravidelně kontroluje seznam neuklizených pokojů. Po dokončení své práce označí pokoj jako uklizený.

V den odjezdu se zákazník dostaví na recepci, kde recepční provede check-out proceduru. Ta zahrnuje změnu stavu rezervace a označení pokoje jako neuklizený.

Dostupné operace

- API/command/clients/
  - [post] - vytvoří klienta
    (v body Client entita)
  - /{id} [delete] - odstraní klienta
  - /{id} [patch] - změní údaje o klientovi
    (v POST body Client entita)
- API/command/reservations/
  - [post] - vytvoří rezervaci a přiřadí ji k zákazníkovi
  - (v body Reservation entita)
  - /{id} [delete] - odstraní rezervaci
  - /{id} [patch] - mění stav a zaplacení rezervace
  - (v POST body ReservationUpdate s atributy State a Paid)
- API/command/rooms/
  - [post] - vytvoří nový pokoj
    (v body Room entita)
  - /{id} [delete] - odstraní pokoj
  - /{id} [patch] - označí pokoj jako (ne)uklizený
    (v POST body RoomUpdate s jediným IsCleaned boolean atributem)
- API/query/clients/
  - [get] - vrátí všechny zákazníky
    (volitelný query parametry Email)
  - /current [get] - vrátí všechny aktuálně ubytované zákazníky
  - /{id} [get] - vrátí zákazníka pokud existuje
- API/query/reservations/
  - [get] - vrátí všechny rezervace
  - /{id} [get] - vrátí jednu rezervaci pokud existuje
  - /client/{id} [get] - vrátí všechny rezervace daného klienta
  - /room/{id} [get] - vrátí všechny rezervace pro daný pokoj
- API/query/rooms
  - [get] - vrátí všechny pokoje
    (volitelné query parametry pro filtraci pro patro, cenu, pocet posteli/přistýlek)
  - /{id} [get] - vrátí jeden pokoj pokud existuje
  - /uncleaned [get] - vrátí všechny neuklizené pokoje
  - /available?from&to [get] - vrátí všechny dostupné pokoje v daném termínu
    (query parametry from/to pro časový interval - pokud nevyplněno, tak k dnešnímu datu - a jinak jako pro normální get)

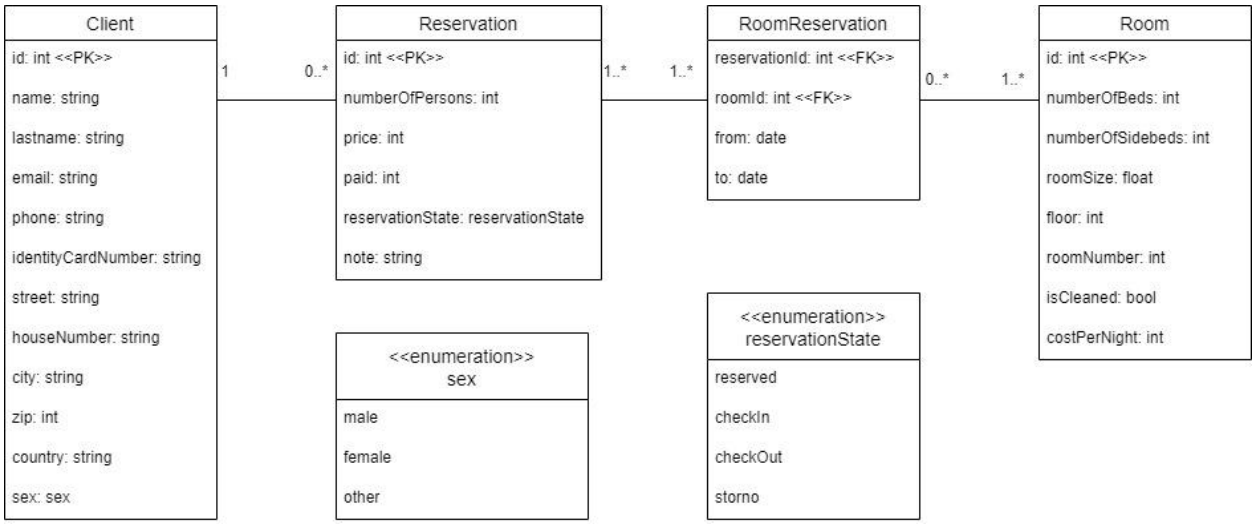
Předmět: PDB
Autoři: Ondřej Studnička (xstudn00), Jan Lorenc (xloren15)
Datum: 4. 12. 2022

Operace ve scénáři nejčastějšího užití

1. Získání dostupných pokojů
   API/query/rooms/available?from={date}&to={date}
2. Ověření existence klienta:
   API/query/clients?query={email}
3. V případě neexistence klienta (nový klient), vytvoření klienta:
   API/command/clients (v POST body budou zákazníkem uvedené informace)
4. Rezervování pokoje
   API/command/reservations (v POST body budou všechna potřebná data pro vytvoření rezervace)
5. Kontrola stavu pokoje (uklizeno)
   API/query/rooms/{id}
6. Potvrzení příjezdu:
   API/command/reservations/{id} (v POST body bude nastaven state na checkIn)
7. Nahlášení požadavku na úklid pokoje
   API/command/rooms/{id} (v POST body bude update s IsCleaned atributem a hodnotou false)
8. Zobrazení neuklizených pokojů uklízečkou
   API/query/rooms/uncleaned
9. Nahlášení úklidu pokoje
   API/command/rooms/{id} (v POST body bude update s IsCleaned atributem a hodnotou true)
10. Potvrzení odjezdu:
    API/command/reservations/{id} (v POST body bude nastaven state na checkOut)
11. Nahlášení požadavku na úklid pokoje
    API/command/rooms/{id} (v POST body bude update s IsCleaned atributem a hodnotou false)

Analýza

Relační databáze se skládá celkem ze 4 tabulek, a sice - Client, Reservation, RoomReservation a Room. Jejich atributy jsou zaznamenány na obrázku níže. Jsou definovány také dva výčtové typy - sex a reservationState.



NoSQL část aplikace uchovává data potřebná k jednotlivým dotazům. Databáze obsahuje následující tabulky:

- clients - všichni klienti … DTO varianta SQL tabulky
  (Recepční pro zobrazení seznamu klientů nepotřebuje znát adresu či pohlaví, kompletní detail se může řešit třeba v jiné části systému. NoSQL nám zde ukládá již projekci.)
- rooms - kopíruje SQL tabulku pokojů, jelikož všechny atributy jsou potřeba k filtraci
- reservationsGroupedByClients - rezervace uložené podle zákazníků
- reservationsGroupedByRooms - rezervace uložené podle pokojů

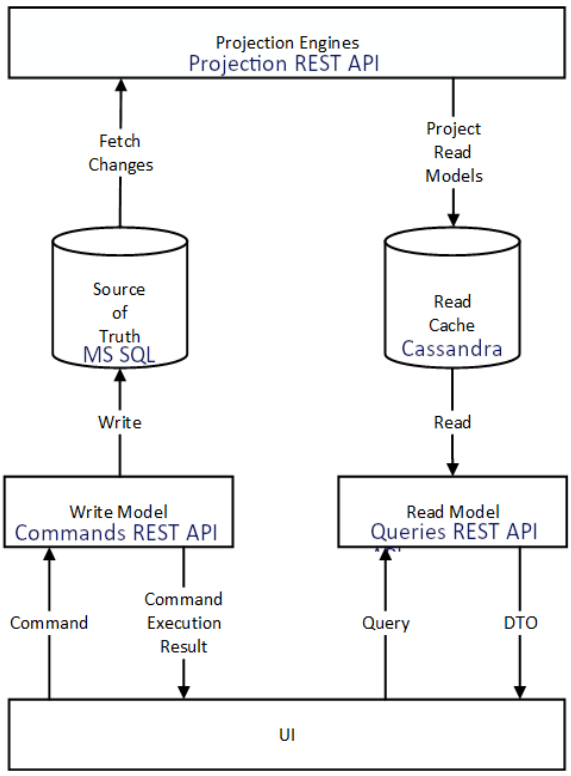
Nad těmito tabulkami je vytvořen i materializovaný pohled:
(Zde ukládáme již předfiltrované údaje k optimalizaci čtecích operací.)

- rooms.uncleaned - obsahuje seznam neuklizených pokojů

Z těchto tabulek/pohledů pak aplikace získává informace pomocí SELECT dotazů. Data se vytvářejí zasíláním Commands, které data zapíší do SQL databáze, a pomocí projection enginu následně proběhne propis/aktualizace dat v NoSQL databázi. Dokud nepřijde požadavek na aktualizaci, lze data považovat za platná. K jiné aktualizaci než skrze Commands nemůže dojít a proto použití Commands určuje rychlost/frekvenci změny NoSQL dat. Pro zvýšení spolehlivosti budou data replikována a existovat budou celkem 3 kopie. Zároveň bude zachována trvalost.

Návrh

Systém bude využívat návrhový vzor CQRS. Commands budou mít za úkol zapisovat relační data do MS SQL databáze, která bude sloužit jako single source of truth. Queries se budou dotazovat na sloupcovou NoSQL databázi Cassandru, která byla zvolena pro její vhodné filtrační vlastnosti. Cassandra shlukuje data podle Partition klíčů, díky čemuž můžeme docílit efektivního dotazování například na rezervace dle klientů či pokojů.



Jelikož se po každém Commandu změní data v SQL databázi, NoSQL data se snadno mohou dostat do nekonzistentního stavu. Tento problém řeší Projection engine, který po každém Commandu data aktualizuje

Každá ze tří částí (Commands, Queries, Projection engine) budou samostatně spustitelná REST API z důvodu rozdělení zodpovědností a škálovatelnosti systému. V budoucnu by totiž mohli přibýt další čtenáři či zapisovatelé, kteří by mohli využívat stejného projection enginu či jinak komunikovat.

Systém bude poměrně snadno testovatelný. Všechna API budou vizualizovaná pomocí Swaggeru a bude možné si všechny dotazy vyzkoušet. Uživatel dle výsledku HTTP požadavku uvidí, zda exekuce proběhla v pořádku, bude si moci prohlédnou vrácená data a například při testování zápisu si rovnou v API pro Queries může otestovat propsání do NoSQL. Sada testovacích url dotazů bude poskytnuta s implementací.

Zvolené technologie

Pro projekt jsme zvolili jazyk C# a platformu .NET. Řešení bude rozděleno na několik projektů v rámci jedné .net solution (.sln soubor pro Visual Studio). Všechna api budou vizualizovaná pomocí Swaggeru a ke vzájemné komunikaci (Commands-Projection) poslouží nswag klient. Single source of truth bude standardní SQL databáze, jíž bude localdb MS SQL serveru. Za NoSQL databázi jsme zvolili Cassandru.

Předmět: PDB
Autoři: Ondřej Studnička (xstudn00), Jan Lorenc (xloren15)
Datum: 4. 12. 2022

Implemenace

Řešení je celé v jedné .NET solution (.sln). Obsahuje 3 spustitelné Web API aplikace, kde každá má vlastní knihovny dle vrstev Clean designu. Všechna API používají Swagger k vizualizaci.

Command API

Tento projekt obsahuje 3 kontroléry pro každou z domén (Clients, Reservations, Rooms). Logika kontrolérů je velmi jednoduchá. Nejprve vyšlou příslušný požadavek mediátoru, jelikož je zde využíván návrhový vzor Mediator, a následně je zavolán patřičný endpoint Projection Enginu. Svou logiku implementuje v knihovně Command.Application, kde jsou implementovány Create|Update|Delete požadavky mediátoru, jenž příchozí data validuje, mapuje z DTO na entity a zapisuje do databáze. Pro validaci je použita knihovna FluentValidation a mediátor samotný je poskytnutý knihovnou MediatR. Databází je relační databáze, kterou je v základu localdb MS SQL serveru, nicméně stačí pozměnit ConnectionString v appsettings.json pro výběr jiné SQL databáze. Pro práci s ní je používán ORM Entity Framework, jehož databázové entity jsou definovány v projektu Hotel.Persistence. Mimo jiné je v tomto projektu i seed databáze a migrace, které jsou zkontrolovány a případně spuštěny při každém spuštění Command API díky StartupFilter funkcionalitě. Není tedy třeba žádného manuálního nastavování databáze. Běží-li SQL server, stačí spustit API a databáze se automaticky vytvoří i se seed daty.

Query API

Poslední z projektů obsahuje rovněž 3 kontrolery (Clients, Rerervations, Rooms). Zde dochází pouze k dotazování NoSql databáze Cassandra, z čehož plyne, že se nejedná o nijak složitý proces. V tomto případě tedy nejprve dojde v každém z kontrolerů k zavolání metody z příslušného repozitáře (návrhový vzor Repository) a získaná odpověď je přemapována na DTO objekt a je vrácena jako výsledek. Pokud se žádná data nevrátí, tak výsledkem bude NotFound(). Logika se nachází v projektu Query.Application. Pro každou doménu jsou v repozitářích implementovány patřičné metody. Scénář je obyčejně takový, že v každé metodě dojde nejprve k navázání spojení s databází. Následně dojde buď rovnou přímo k dotazování databáze, případně k připravení dotazu a pak k dotazování. Získaná odpověď je zpracována pomocí RowSetMapper a jeho metod. RowSetMapper mapuje získaná data na modely, které jsou rovněž definovány v Query.Application. Pro vytvoření keyspace, potřebných tabulek a materializovaného pohledu stačí spustit projekt Query. Pokud keyspace nebo tabulky neexistují, tak dojde automaticky k jejich vytvoření. Jejich struktura je definovaná v Hotel.Persistence/Cassandra/HotelContextCassandra.cs.

Projection Engine API

ProjectionEngine obsahuje jeden kontroler, který zabezpečuje přenos dat z relační do NoSql databáze pro pokoje, rezervace i klienty. Definuje endpointy pro přenos nových/aktualizovaných dat i pro smazání požadovaných. Rovněž jako v případě Query části aplikace je využíván návrhový vzor repozitář. Logika je tedy opět implementována v něm. Zavolá se patřičná metoda z repozitáře a v metodě dojde k uplatnění logiky samotné. V rámci repozitáře jsou definovány metody i společně s databázovými dotazy. Scénář je velmi podobný jako výše, tedy vytvoření spojení s databází a následně k přípravě dotazu a jeho vykonání. Tyto metody nevrací žádnou odpověď. Pokud se vyskytne nějaká chyba, tak je odchycena v kontroleru a je vrácen její popis.

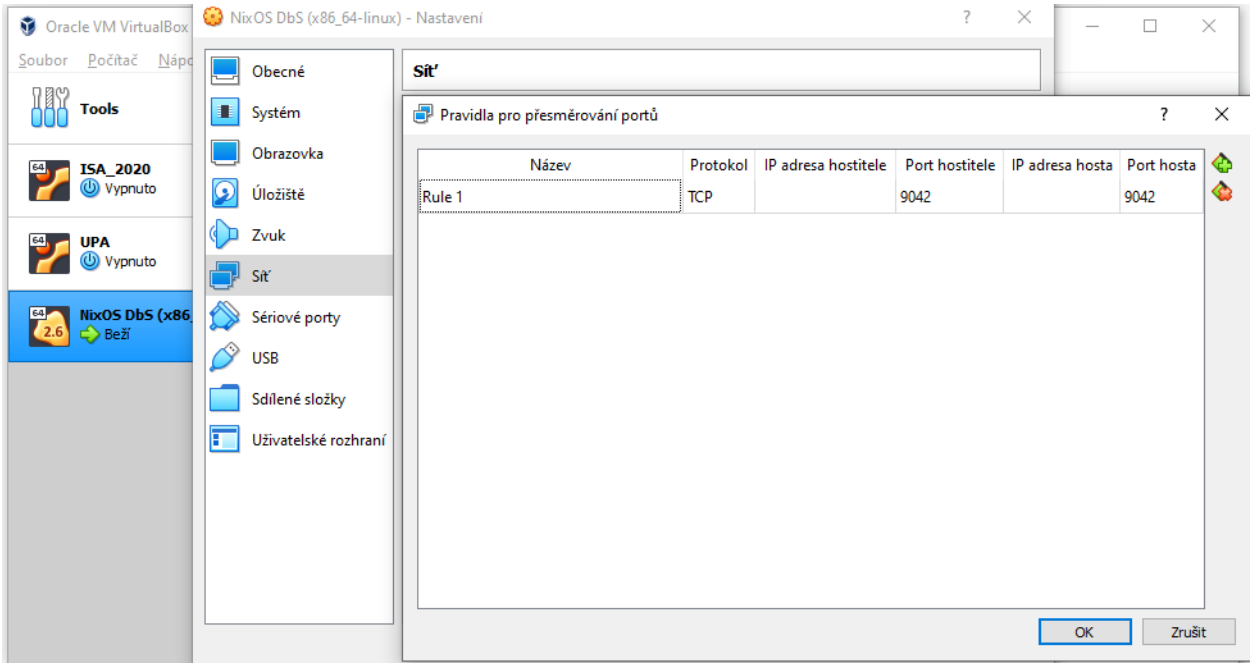
Spuštění

Relační databáze

V první řadě je třeba mít k dispozici MS SQL Server a zde vytvořenou databázi s názvem MSSQLLocalDB. Následně stačí příkazem `sqllocaldb start MSSQLLocalDB` spustit a vše je připraveno.

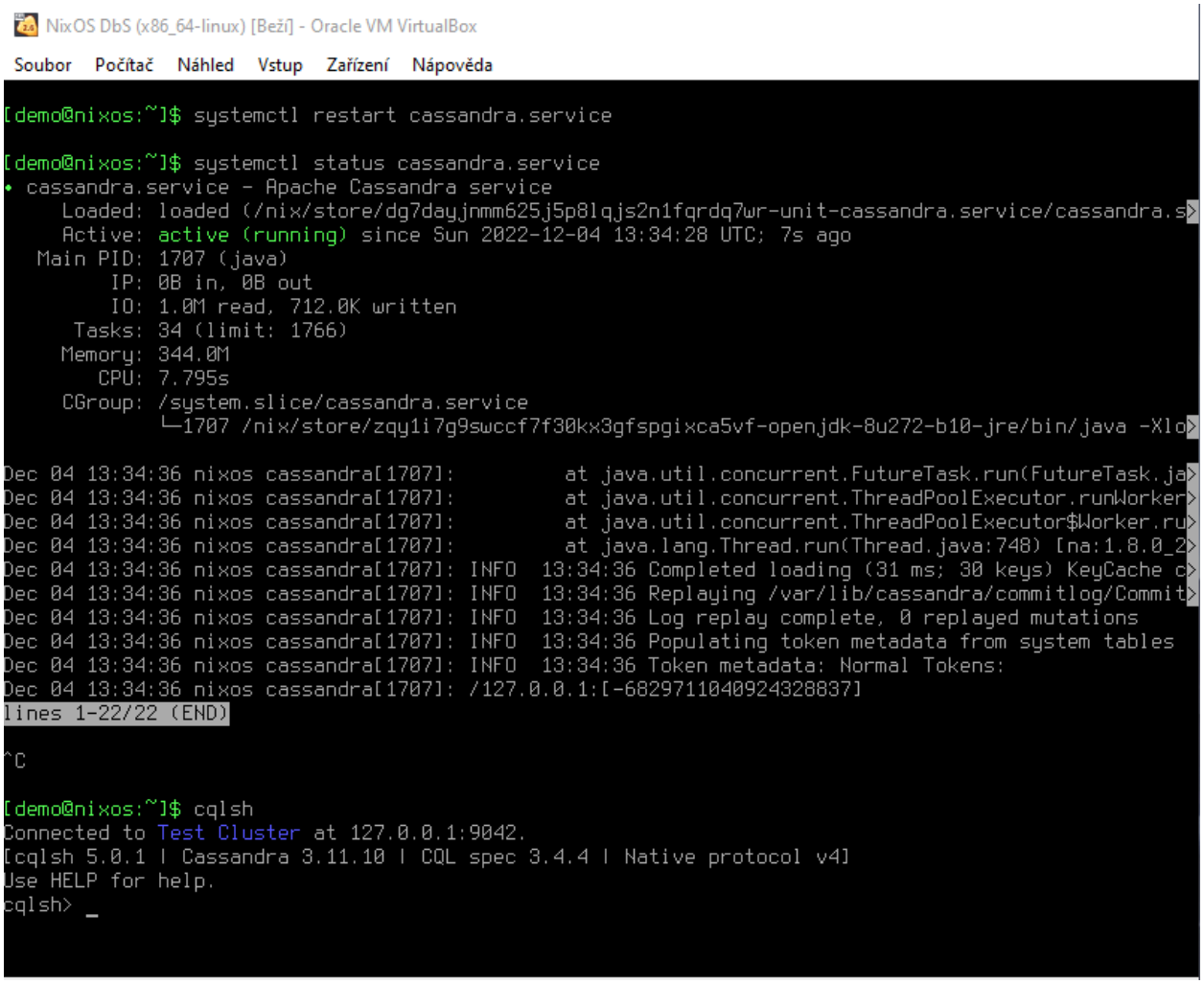
NoSQL databáze

V tomto případě je třeba stáhnout obraz virtuálního počítače z https://rychly-edu.gitlab.io/dbs/nosql/nixos-dbs-vm a provést import appliance. Následně je potřeba nastavit předávání portů (Nastavení -> Síť -> Pokročilé -> Předávání portů) . Port hosta i hostitele bude nastaven na 9042.



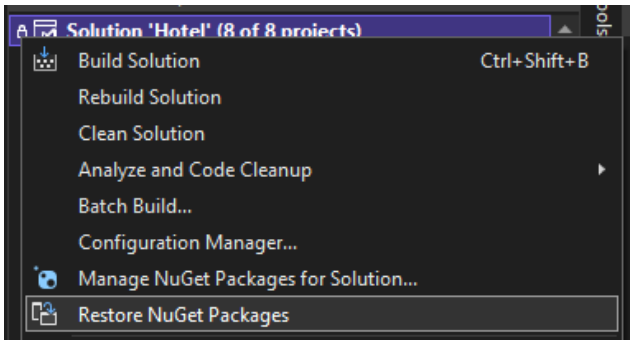
Následně je třeba spustit počítač a po jeho zapnutí zadat zadat příkazy `systemctl restart cassandra.service` a `systemctl status cassandra.service`.

Po úspěšném provedení těchto dílčích částí by se měl uživatel dostat do stavu na obrázku.

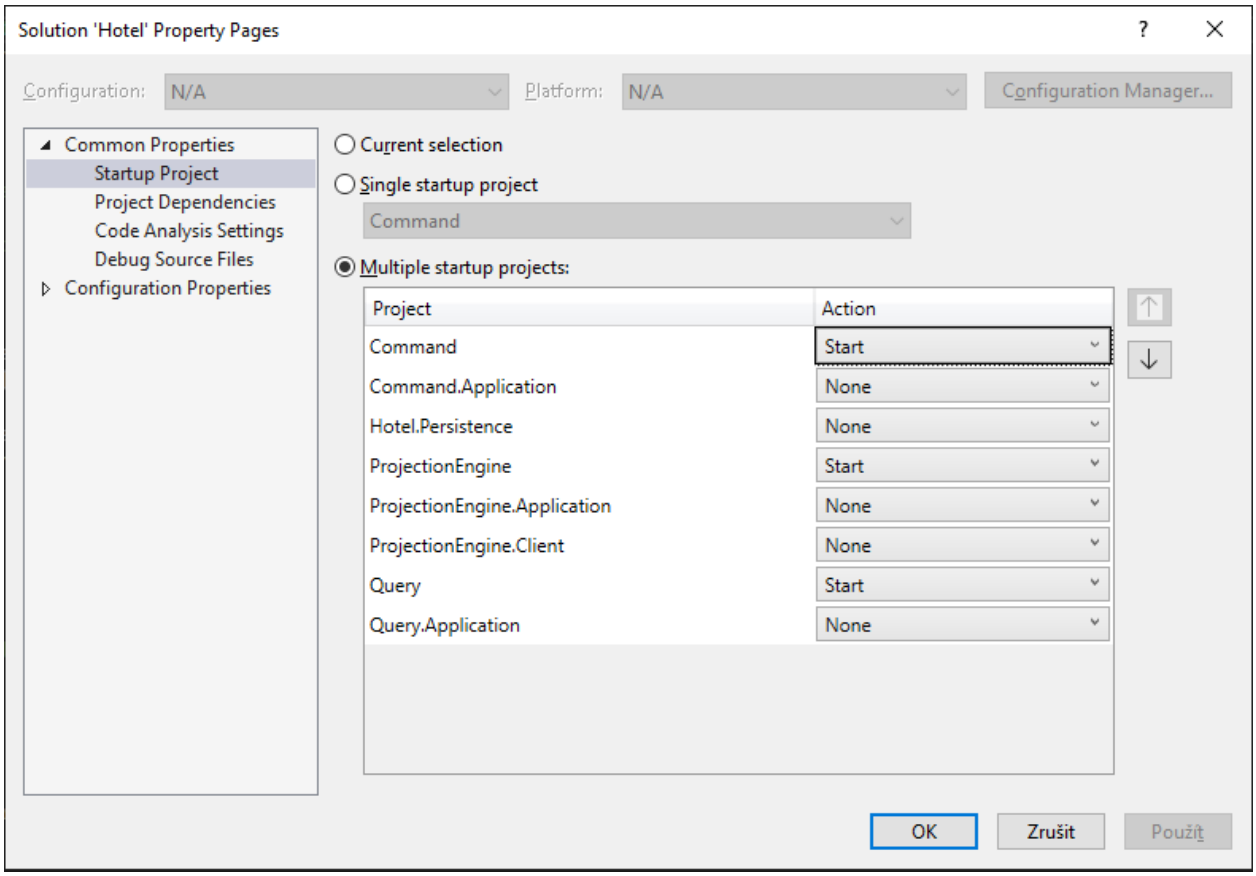


Spuštění projektu

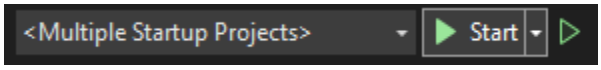
Pro spuštění projektu je ideální mít k dispozici Visual Studio 2022. Ve složce src se nachází soubor Hotel.sln. Ten je třeba otevřít. Po otevření najdeme v Solution Explorer “Solution Hotel”. Nad ním pravým tlačítkem otevřeme menu a vybereme “Restore NuGet Packages”. Tím by mělo být zabezpečeno stažení všech potřebných závislostí.



Následně je třeba ještě nastavit projekty pro spuštění. To uděláme tak, že opět otevřeme menu nad “Solution Hotel” a v nabídce vybereme Properties. V záložce Startup Project zvolíme možnost Multiple startup projects a zde pro Command, ProjectionEngine a Query nastavíme Action na hodnotu Start.



Tím je nastavení dokončeno a stačí už jen samotné spuštění.



Při prvním spuštění je dobré (ne však nutné, pokud před využitím Query bude v Commands vytvořen nový pokoj, rezervace a klient) provolat endpointy projection-engine/clients, projection-engine/reservations, projection-engine/rooms. Tím dojde k projekci dat z relační do NoSql databáze.

Automatické spuštění

Druhou možností, jak spustit projekt je použití scriptu `restore_build_run.bat`. Ten provede výše popsané kroky a provede i projekci z relační do nerelační databáze. Spuštění výše popsaným způsobem je jistější.

Předmět: PDB
Autoři: Ondřej Studnička (xstudn00), Jan Lorenc (xloren15)
Datum: 4. 12. 2022

Testování

CRUD operace nad každou doménou byly manuálně testovány ve Swaggeru. V Command API se vytvořila entita a pomocí Query API se vyhledala a tím se ověřilo, že se vše uložilo a skrze Projection Engine i propsalo do NoSQL. Stejným způsobem byla provedena operace Update a to nad každým atributem třídy. U obou (Create i Update) jsme ověřili i funkčnost validace zadáním prázdných povinných atributů nebo vytvářením již existující či upravování neexistující entity. Propsání do NoSQL jsme opět otestovali získáním z Query API. Podobně se postupovalo i pro Delete operaci. Mimo to jsme korektní provolávání API ověřili i přes debugging. Takto tedy byly otestovány všechny endpointy. Na závěr jsme úspěšně vyzkoušeli uvedený scénář nejčastějšího užití.