UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI

**FACULTATEA DE INFORMATICĂ**



LUCRARE DE LICENȚĂ

**Avantajele patternului Event Sourcing**

**propusă de**

***Gabriel-Angelo Panțiru***

**Sesiunea:** *iulie, 2018*

**Coordonator Științific**

**Colab. Florin Olariu**

**UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IASI**

**FACULTATEA DE INFORMATICA**

**Avantajele patternului Event Sourcing**

***Gabriel-Angelo Panțiru***

**Sesiunea:** *iulie, 2018*

**Coordonator Științific**

***Colab. Florin Olariu***

DECLARAŢIE PRIVIND ORIGINALITATE ŞI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licenţă cu titlul „*Avantajele patternului Event Sourcing*” este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învăţământ superior din ţară sau din străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

* toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi deţin referinţa precisă a sursei;
* reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa precisă;
* codul sursă, imaginile etc. preluate din proiecte *open*-*source* sau alte surse sunt utilizate cu respectarea drepturilor de autor şi deţin referinţe precise;
* rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iaşi,

Absolvent *Gabriel-Angelo Panțiru*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original)

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Avantajele patternului Event Sourcing*”, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date detest etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi,

Absolvent *Gabriel-Angelo Panțiru*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original)

Introducere

Am ales aceasta tema pentru lucrarea de licenta deoarece Event Sourcing este un pattern care poate oferi foarte multe beneficii. In cele mai multe aplicatii este suficient sa putem interoga baza de data si sa aflam ultima stare a ei. Sunt insa multe cazuri unde ar fi avantajos sa salvam toate starile prin care a trecut aplicatia deoarece acest lucru poate oferi posibilitati de analiza a modului cum se schimba starea aplicatiei ceea ce ar putea aduce beneficii aplicatiilor din anumite domenii cum ar fi cel economic. Pentru a ilustra aceste avantaje am ales sa implementez o aplicatie folosind acest pattern oferind in acest fel explicatii sa le leg de scenarii reale de folosire.

“Event Sourcing este o modalitate de a persista starea aplicatiei tale folosind, stocand o istorie din care se poate determina starea curenta a aplicatiei“[1] .

Patternul de event sourcing este folosit frecvent impreuna cu CQRS si acesta poate fi benefic deoarece permite separarea modelului de citire si modelului de scriere, oferind in acest mod posibilitatea de a lucra la eficienta operatiilor de citire si a operatiilor de scriere in mod separat, lucru care este util in aplicatiile in care se fac foarte multe operatii de citire dar putine de scriere sau invers si este necesara eficientizarea doar a unui subset de operatii care sunt mai importate pentru business.

In event sourcing un agregat reprezinta entitatea de care sunt legate un set de evenimente si fiecare eveniment legat de acel agregat reprezinta o versiune a entitatii.

Unul alt beneficiu al acestei arhitecturi este ca aceste evenimente pot fi folosite pentru restaurarea starii aplicatiei la un anumit moment din trecut.

De asemenea evenimentele pot fi procesate oricand de consumere si folosite in scopuri precum: reconstituirea ultimei stari a aplicatiei, analiza modului in care starile aplicatiei se schimba, determinarea celor mai folosite functionalitati (cu cat sunt mai folosite cu atat prezinta un interes mai ridicat pentru stakeholders).

Sisteme distribuite

Sistemele distribuite de software reprezinta o multime de componente software care comunica intre ele pentru a atinge un scop comun. Beneficiile folosirii sistemelor distribuite constau in capacitatea de specializare a componentelor individuale asupra tipului de task pe care trebuie sa il indeplineasca. In sistemele distribuite aplicatiile sunt impartite in componente mai mici lucru care incurajeaza reutilizarea componentelor deja existente si imbunatateste in acest mod procesul de dezvoltare al softwarelui.

Event Sourcing in sisteme distribuite

Avand in vedere ca sistemele distribuite se bazeaza pe comunicarea intre componentele din care sunt formate, event sourcing poate fi un pattern foarte potrivit pentru a implementa in aceste sisteme. Cand o componenta din un sistem distribuit trebuie sa ii transmita informatii unei alte componente, acest lucru se poate face prin publicarea si consumarea de evenimente. Daca evenimentele sunt salvate in o baza de date ar putea constitui un avantaj deoarece in acest mod daca o componenta nu functioneaza pentru o perioada de timp, aceasta poate parcurge evenimentele pierdute si ajunge la starea la care trebuia sa fie daca ar fi functionat tot timpul.

Framework-ul CQRSlite

In implementarea aplicatiei am am folosit framework-ul CQRSlite care ofera un set de interfete si functionalitati comune pentru a usura munca implementarii unei aplicatii prin acest pattern. Acest framework pune la dispozitie utilizatorului functionalitati de trimitere comenzi si publicare de evenimente,posibilitatea de a lucra cu snapshoturi si rezolva problema concurentei adaugand in mod automat un time stamp si eu versiune fiecarui eveniment. In patternul de event sourcing evenimentele sunt procesate pentru a ajunge la ultima stare a aplicatiei, lucru care ar putea dura destul de mult cand sunt mult evenimente de procesat iar pentru a rezolva problema din aceasta situatie intervine conceptul de snapshotting. Prin snapshoting se salveaza o anumita stare intermerdiara a aplicatiei iar evenimentele se proceseaza doar de la acel snapshot inainte, marind in acest fel viteza aplicatiei atunci cand sunt foarte multe evenimente de procesat.

Analiza datelor din evenimente

Daca aceste evenimente sunt persistate intr-o baza de date atunci pot fi folisite pentru a analiza si crea diferite statistici referitoare la modul in care aplicatia e folosita sau asupra datelor care sunt introduse in ea. Aceste evenimente din baza de date pot fi vazute ca o istorie completa a tuturor modificarilor asupra datelor din aplicatie, nu doar ultima stare a datelor, ca la majoritatea aplicatiilor, iar acest lucru permite crearea unor statistici complexe care sa ofere o viziune mai buna asupra domeniului aplicatiei, cum e folosita si cum ar putea fi imbunatatita pentru a maximiza serviciile pe care aceasta le ofera.

Contributie personala

In aceasta lucrare de licenta voi urmari sa analizez sistemele distribuite, modul in care aceste sisteme pot eficientiza procesul dezvoltarii software si avantajele folosirii event sourcing-ului in aceste sisteme. Pentru a indeplini acest lucru voi implementa o aplicatie folosind aceasta arhitectura si o voi folosi ca studiu de caz pentru a gasi avantajele dar si avantajele utilizarii acestei arhitecturi. De asemenea voi trage concluzii din aceast studiu de caz, tipurile de aplicatii pentru care aceasta arhitectura este cea mai potrivita.

Tipuri de aplicatii potrivite pentru aceasta arhitectura

Deoarece aceasta arhitectura poate aduce in unele situatii avantaje iar in altele dezavantaje, ea este folosita pentru tipurile de aplicatii in care avantajele intrec dezavantajele. Dintre aplicatiile carora aceasta arhitectura aduce beneficii fac parte: cele care au nevoie de scalabilitate mare, cele care au nevoie de un istoric complet al modificarilor aici intrand aplicatiile de tip economic ca cele de contabilitate sau sistem de gestionare al bancilor unde fiecare tranzactie trebuie sa fie stocata din motive de securitate.

Tipuri de aplicatii nepotrivite pentru aceasta arhitectura

Aceasta arhitectura odata implementata vine cu o anumita serie de beneficii, insa sunt aplicatii in care aceste beneficii nu isi au rostul sau ar fi prea mici ca sa justifice efortul implementarii arhitecturii distribuite. In aceasta categorie intra aplicatiile de dimensiuni mici, care nu au nevoie de multe resurse si nu sunt proiectate pentru a fi scalabile, cele care nu pot fi distribuite din cauza ca fac parte din sisteme hardware de sine statatoare cum ar fi softul ce controleaza liftul sau cel ce controleaza partea electrica dintr-o masina.

Prezentarea aplicatiei

Scopul aplicatiei va fi sa faciliteze studiul pentru materiile de la facultate. Aplicatia este formata din 2 module principale. Un modul de time management unde poti stabili timpul pe care il vei dedica activitatilor de studiu prin creearea de taskuri cum ar fi "Implementeaza o aplicatie cu arhitectura distribuita", la care poti aloca un anumit numar de ore si pe care le poti seta ca terminate sau poti actualiza numarul ramas de ore pentru lucrul la acel task. Modulul de time management este by default public pentru ca utilizatorii sa se poata inspira unii de la altii in legatura cu programul de studiu. Celalalt modul va fi unul in care sa se poata crea topici de discutii sau intrebari ce pot fi marcate "answered" de catre cel care a pus intrebarea cand primeste un raspuns satisfacator. Ceea ce e nou la aceasta idee fata de grupurile de pe facebook, ca FII 2014-2017,este ca totul ar fi intr-un loc si vor fi in mare parte doar informatii legate de facultate, lucru care ar putea faciliza cautarea anumitor concepte specifice. Plus ca avand doar un singur scop ar fi mai usor sa te concentrezi pe studiat, spre deosebire de facebook unde este greu ca atentia sa nu iti fie distrasa de postarile sau mesajele prietenilor.

Detalii despre implementarea aplicatiei

1. Proiectul WebApi

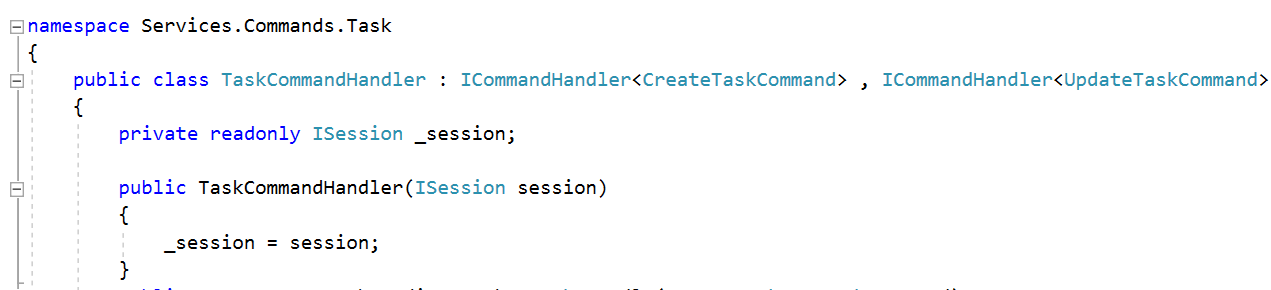
Proiectul WebApi este de tipul ASP.NET CORE MVC. In el sunt controllerele si viewurile. Desi modelele folosite pentru comunicarea intre viewuri si controllere sunt in mod normal in acest proiect, in cazul de fata sunt in proiectul de Servicii deoarece arhitectura CQRS permite ca serviciile sa intoarca doar strictul necesar de care au nevoie viewruile, nemaifiind nevoie de modele in proiectul WebApi care sa filtreze informatiile primite de la servicii. De asemenea in acest proiect sunt inregistrate si configurate majoritatea functionalitatilor de baza din .net core sau pachetele nuget ca Dependency Injection sau libraria Asp.Net Identity folosita la autentificare. Modele folosite pentru comunicarea intre controllere si viewuri sunt respecta patternul de CQRS, modelele pentru post/update/delete derivand din clasa ICommand iar modelele de rezultat pentru query derivand din clasa IQueryResult. Patternul CQRS are avantajul ca permite simplificarea obectelor prin care se transfera date intre controller si view folosind cate un obiect separat pentru fiecare request, in acest fel fiecare putand avea strict datale necesare requestului, reteaua nefiind incarcata cu proprietati care ar fi fost utile altor requesturi. In viewrile razor am folosit frameworkul Bootstrap de css pentru stilizare si cod cshtml pentru redactea lor. Tot in acest proiect se realizeaza si procesul de autentificare care e implementata folosind libraria Identity, o solutie ce ofera implementari pentru majoritatea functionalitatilor de autentificare si autorizare si care poate fi folosita doar prin adaugarea unor setari de configurare in fisierul Startup.cs.

1. Proiectul Services

In proiectul Services se face separarea intre comenzi si query-uri el fiind impartit in doua subfoldere „Commands” si „Queries”. In Commands se gasesc doua subfoldere Task si Topic care reprezinta entitatile de baza cu care lucreaza aplicatia. In fiecare din aceste doua se gasesc implementari similare doar proprietatile si diferitele actiuni care se pot aplica fiind diferite. In Task gasim comenzile si hadlerele ce prelucreaza aceste comenzi:

* ChangeTaskStatusCommand - comanda pentru marcarea unui task ca fiind completat
* CreateTaskCommand – comanda ce contine datele initiale despre un anumit task
* UpdateTaskCommand – comanda ce contine noua versiune a descrierii sau titulului task-ului
* TaskCommandHander – clasa ce are metode „Handle” care pot primi ca parametri comenzi legate de entitatea task

Comenzile deriveaza din interfata marker „ICommand” care nu contine proprietati sau metode iar TaskCommandHandler deriva din mai multe interfete IcommandHandller<Command>, obligand clasa sa implementeze metoda Handle(Command command). Din aceasta functie comanda este transformata in un eveniment si transmisa la EventStore, in cazul in care comanda este pentru creare unei entitati noi sau datele din comanda sunt pasate catre o functie a clasei aggregatului de care este legat si acolo este transformata in eveniment si transmisa la EventStore pentru a fi inserata in baza de date. De asemenea in functia Handle de la command handler se pot face anumite verificari pentru a decide daca evenimentul poate fi creat, un exemplu ar fi sa verifici daca un task exista inainte de a a-l marca ca fiind terminat.



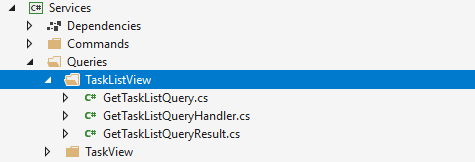
In subfolderul Topic de la Commands gasim urmatoarele comenzi si handlere:

* CreateTopicCommand - folosita la crearea unui nou topic; contine titlul si continutul acestuia
* UpdateTopicCommand – contine date similare cu comanda pentru crearea unui nou topic, insa se poate aplica numai daca topical nu contine deja raspunsuri
* AddNewReplyCommand – contine raspunsul, cine l-a oferit si data cand a facut-o
* UpdateReplyCommand – similara cu comanda de adaugare a unui raspuns dar trebuie sa fie verificat daca acel reply exista deja inainte de a se crea evenimentul de ReplyUpdated
* TopicCommandHandler – la fel ca la command handler-ul pentru taskuri, acesta are metode „Handle” care pot primi ca parametri comenzi legate de entitatea topic

Tot in proiectul Services avem foldrerul querries care este organizat in subfoldere pe urmatoarele viewuri :

* TaskListView – acest view contine informatiile afisate in pagina cu lista de taskuri cum ar fi Id-ul, titlul sau daca taskul e terminat sau nu
* TaskView – contine toate detaliile legate de un task si este folosit pe pagina de task details
* TopicListView – contine informatiile necesare pentru pagina listei de topicuri
* TopicView – contine toate informatiile unui topic : titlu, descriere si raspunsuri

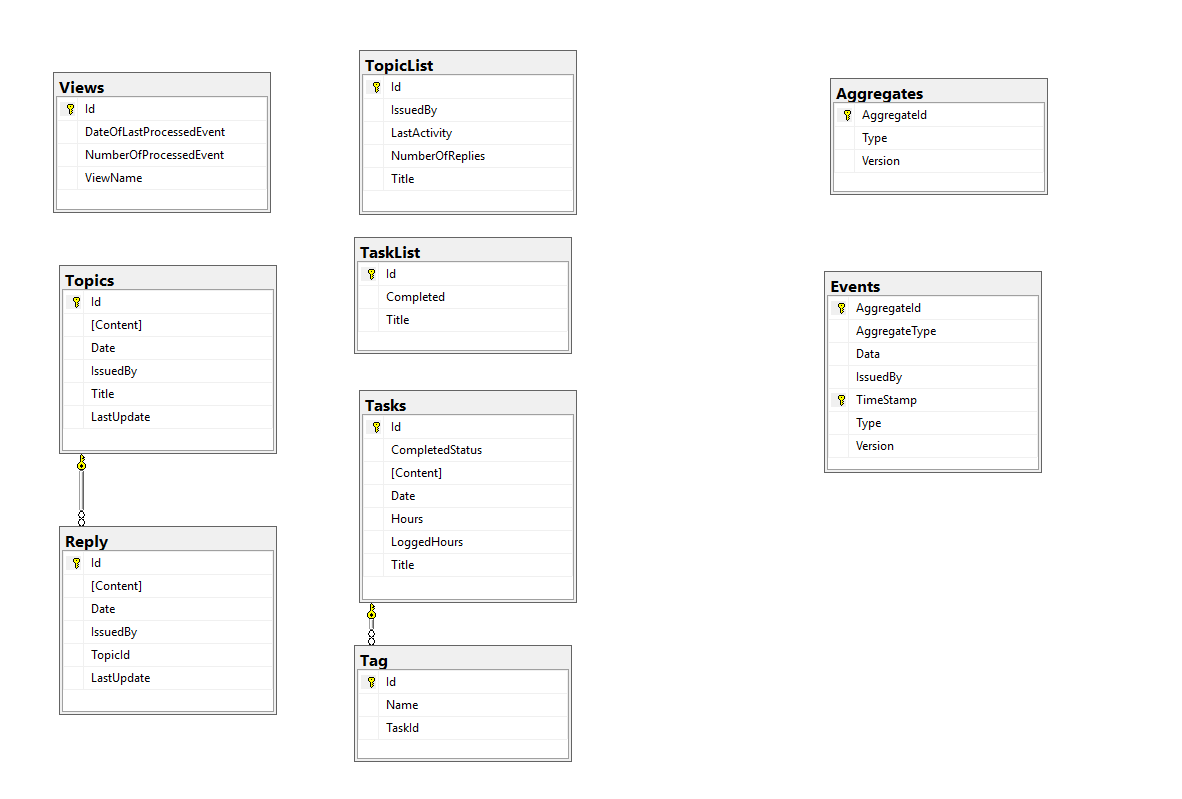
Datele din aceste viewuri sunt completate de catre aplicatia ViewProcessorConsumer pe masura ce proceseaza noile evenimente. In fiecare dintre subfulderele descrise mai sus se gaseste o clasa care contine informatiile pe baza carra se face query-ul (de obicei id), o clasa care reprezinta modelul returnat de query si clasa handler care face cererea pentru respectivele date la repository. Cele 3 clase descrise mai sus deriva din IQuery, IQueryHandler si IQueryResult.



1. View Processor

In proiectul ViewProcessor se gasesc handlerele pentru toate viewurile. In aceasta aplicatie un view este diferit de un viewurile din proiectul WebApi unde descriu browserului prin cshtml cum sa arate pagina. In acest proiect un view reprezinta o anumita tabela in baza de date care contine datele necesare de afisat pe o anumita pagina. De exemplu pagina unde se afiseaza lista de taskuri are nevoie doar de titlul taskului, id-ul si daca este completat sau nu. Un handler trateaza orice tip de eveniment in functie de modul in care acel eveniment afecteaza acel view; de exemplu handlerul pentru view-ul TopicList isi va updata continutul pentru evenimente de tipul TopicUpdated insa nu se va modifica pentru evenimentul TaskUpdated deoarece acel eveniment nu schimba cu nimic continutul datelor din viewul TopicList.

Baza de date



Pentru a ilustra mai bine conceptul de sisteme distribuite in aplicatia mea evenimentele le voi salva in o baza de date iar viewurile le voi materializa in o baza de date separata. Aceasta distributie poate aduce avantaje foarte interesante deoarece in acest mod baza de date unde se insereaza evenimente poate fi una optimizata pentru inserari, de exemplu non relationala, iar baza de data ce contine tabelele de viewuri poate fi una optimizata pentru citiri, cum ar fi cea de tip relationala. Dar aceasta arhitectura de folosire a doua baze de date pentru a eficientiza partea de read si partea de write ar putea aduce si dezavantaje deoarece acest lucru ar insemna ca aplicatia trebuie sa ia informatii din prima baza de date si sa le foloseasca in a doua, lucru care ar putea fi destul de costisitor din punct de vedere al timpului.

Data Layer

Proiectul DataLayer contine clasele ce interactioneaza cu bazele de date. Baza de date pentru viewuri a fost generata folosind modul de lucru Code First din Entity Framework si in acest proiect gasim Contextul acestei baze de date, repository-ul si migrarile care au fost folosite la updatarea tabelelor. Tot in acest proiect avem si EventStore, care este o clasa ce deriva din interfata IEventStore gasita in framework-ul CQRSlite, care se ocupa cu inserarea sau citirea evenimentelor in baza de date. Tabelele in care sunt salvate informatiile despre evenimente sunt:

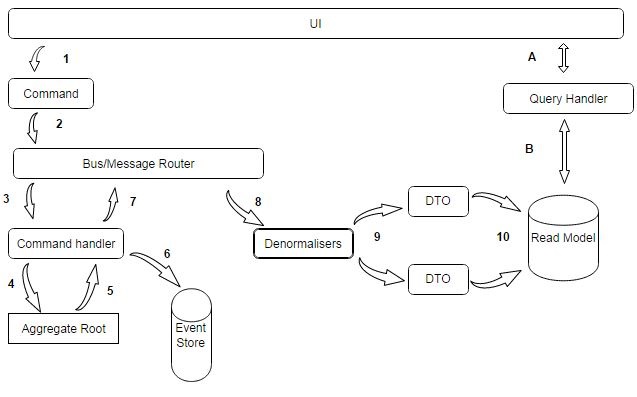
* Aggregates, unde se salveaza Id-ul aggregatului,versiunea acestuia (reprezentand numarul de evenimente pentru acel aggregat) si tipul acestuia, care in aplicatia implementata de mine poate fi Task sau Topic
* Events, in aceasta tabela se salveaza toate detaliile legate de un eveniment precum AggregateId, TimeStamp, IssuedBy, Type, Version si Data unde se salveaza toate informatiile specifice acelui eveniment

Cand se adauga un nou eveniment in baza de date acesta operatie necesita actualizarea atat a tabelului Aggregates cat si Events ceea ce necesita ca aceasta operatie de adaugare a unui nou eveniment sa fie tranzactionala. Pentru procesul de inserare a noilor evenimente clasa EventStore este punctul de lucru cu baza de date. De asemenea cu clasa EventStore interactioneaza alte clase din framework-ul CQRSlite, cum ar fi Sesiunea (obiect intotalitate implementat de catre framework) sau obiectele ce deriva din clasa abstracta AggregateRoot.

1. Proiectul Domain

In proiectul Domain gasim definite obiectele de baza ale aplicatiei, care sunt folosite in toate celelalte proiecte. Aceste obiecte sunt organizate in 3 subfoldere: Aggregates, Events si Views. In aggregates sunt obiectele ce deriva din clasa abstracta AggregateRoot din framework-ul CQRSlite. Clasa AggregateRoot este in primul rand folosita ca marker, alte clase din framework lucrand cu ea, iar in al doilea rand contine functia ApplyEvent care primeste ca parametru un eveniment legat de acel agregat si il insereaza in baza de data folosindu-se de EventStore. In implementarea proiectului meu clasele Task si Topic deriva din ea. In folderul Events se gasesc toate evenimentele folosite in aplicatie, atat pentru topic cat si pentru task. In Views se gasesc toate modelele care se folosesc in application context si dupa care este creata baza de date. Desi clasele Task si Topic se gasesc si in Aggregates si in Views, cele din Views nu deriva din clasa abstracta AggregateRoot, si sunt folosite pentru functionalitati diferite.

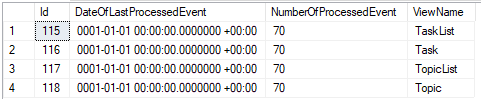
Flow-ul aplicatiei la partea de write



Din UI se trmite o comanda catre back-end cand un utilizator ia o actiune sa modifice starea aplicatiei. Aceasta ajunge in controller si de acolo este pasata unui obiect de tipul message router care stie ce fel de command handler sa instantieze care sa se ocupe de prelucrarea acelei comenzi. Ajunsa in command handler, asupra comenzii se fac verificari care sa determine daca se poate genera un eveniment din ea. Daca conditiile se indeplinesc atunci, in cazul in care comanda este legata de o entitate deja existenta in baza de date, informatiile din comanda sunt trimise catre clasa ce implementeaza AggregateRoot pentru acea entitate si acolo este generat evenimentul sau daca comanda contine informatii despre crearea unei noi entitati, evenimentul este generat direct in command handler si introdus in Event Store prin sesiune.

Flow-ul aplicatie la partea de read

Odata inserat in tabela de evenimente, acesta e gata sa fie procesat de aplicatia EventsConsummer care interogheaza constant baza de date cu evenimente si le proceseaza pe cele noi. Aplicatia EventsPrcessor stie care evenimente au fost procesate si care nu, pentru fiecare view, prin tabela "Views" unde memoreaza numele view-ului, numarul evenimentelor procesate pentru acel view si o data pentru cand a fost procesat ultima oara un eveniment pentru acel view. Pentru a verifica daca trebuie sa actualizeze informatiile in view-uri, aplicaia numara numerul de evenimente din tabela "Events" si compara cu numarul de evenimente procesate din tablea "Views", iar daca gaseste o diferenta atunci incarca in memorie ultimele evenimente neprocesate si apeleaza handler-ul potrivit pentru a extrage informatiile din eveniment si actualiza tabela respectiva. Cand un utilizator vrea sa acceseze niste date din viewuri, din front-end se trimite un query catre controllerem de unde este pasat la un query bus care stie sa instantieze query handler-ul potrivit pentru a se ocupa ce cerere. In Query handler este interogat repository-ul iar datele sunt pasate inapoi catre controller pentru a putea fi afisate utilizatorului



Salvarea contra broadcastul evenimentelor

Sunt mai multe moduri de a implementa o aplicatie ce are la baza event sourcing, fiecare mod putand avea avantajele lui, iar pentru a alege modul de implementare potrivit trebuie avut in vedere necesitatile aplicatiei. In continuare voi discuta doua dintre aceste moduri si explica motivul pentru care am ales sa implementez modul de salvare a evenimentelor. Cand se foloseste modul broadcast evenimentele sunt transmise componentelor responsabile cu procesarea lor, fara a fi insa salvate in baza de date, iar dezavantajul acestui mod de lucru este ca daca o componenta are o eroare sau serverul pe care ruleaza este down, atunci acea componenta va pierde informatiile din evenimente si nu va mai putea reprezenta ultima stare a informatiilor din aplicatie. Pe de alta parte modul de lucru prin salvarea evenimentelor are avantajul ca componentele pot ajunge intotdeauna la ultima stare a aplicatiei deoarece daca serverul este down pentru o perioada, aceasta poate interoga baza de date si procesa toate evenimentele pierdure in timpul cand a fost down. Desigur fiecare mod poate avea avantajele sale in diferite aplicatii, de exemplu daca istoricul din aplicatie nu este important, atunci ar fi mai avantajos de implementat modelul de broadcast al evenimentelor deoarece in acest mod nu s-ar mai irosi resurse.

Bibliografie

1. <https://msdn.microsoft.com/en-us/library/jj591559.aspx>
2. <https://exceptionnotfound.net/real-world-cqrs-es-with-asp-net-and-redis-part-1-overview/>
3. https://www.codeproject.com/articles/991648/cqrs-a-cross-examination-of-how-it-works