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

Proiectul WebApi este de tipul ASP.NET CORE MVC. In el sunt controllerele, viewurile si modelele. 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

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