Costina Horia

Anul 4, IS, Gr. 2

Tema 2 PSSC

Domain-Driven Design: Repository

Domain-Driven Design este o metoda de a transforma o problema complexa intr-un model ce poate fi implementat ulterior de echipa de dezvoltare software.

Modelele DDD sunt definite de un numar de concept high-level ce pot fi folosite atat pentru creere cat si pentru modificare. Aceste modele sunt:

- Entity – un obiect cu identitate unica ce este definit de atribute, dar opus fata de un obiect traditional.
- Value Object – un obiect ce are atribute si nu poate fi schimbat, dar nu are o identitate distincta
- Domain Event – un obiect ce este folosit ca sa inregistreze un event relationat la o activitate model intr-un system.
- Aggregate – un cluster de entitati si value objects cu limite definite in jurul grupului.
- Service – o operatie sau o forma de logica business ce nu se potriveste natural fara un domeniu al obiectelor.
- **Repositories –** difera fata de o versiune de repository de control traditional, ci este mai mult un serviciu ce stocheaza entitati si value objects.
- Factories – incapsuleaza logica creearii complex a obiectelor si a agregatelor asigurandu-se ca un client nu are nicio cunostinta de manipulare interna a obiectelor.

Repository:

In acest caz Repositories se refera la un serviciu ce foloseste o interfata globala pentru a da acces la toate entitatile si obiectele valoare ce fac parte din anumite colectii. Acestea joaca un rol important deoarece fac legatura dintre layerele de domeniu si de mapare a datelor. Totodata, fiind ca o colectie, trebuie sa ofere posibilitatea de a adauga, obtine si eventual sterge obiecte din interiorul lui.

Implementarea concreta se bazeaza pe faptul ca putem stoca obiecte in baze de date relationate, in documente cu baze de date, toate acestea aflandu-se in layerul de infrastructura al domeniului. O implementare simpla ar fi implementarea unei memorii ce doar tine obiecte stocate pe o perioada nelimitata fara a le modifica. Aceasta metoda este simplu de testat deoarece se pot crea usor atat teste unitare cat si module teste. In cazul in care se doreste trecerea la o baza de date atat implementarea cat si testele nu sunt greu de adaptat.

Pentru implementarea unui Repository au fost definite mai multe patternuri. Interfata generica a acestuia este codata pentru tipuri individuale de entitati. Daca doresti ca un repository sa fie pentru un tip specific de entiate trebuie doar sa creezi o derivate aproape de tipul interfetei cu entitatea.

Pentru a obtine un obiect sau o entitate este necesara folosirea unei metode specializate de tipul Query (o metoda specifica de cautare). Aceste metode, de regula, sunt implementate cu ajutorul template-urilor deoarece ele trebuie sa fie cat mai generice si sigure din punctul de vedere al tipului entitatilor.

Clasele ce folosesc un Repository rareori folosesc toate metodele din acestea. De exemplu o clasa ce afiseaza o informatie din acesta clientului nu va folosi si metoda de stergere.

Un Repository o data implementat nu este necesar sa fie schimbat in cazul in care se doreste a fi folosit si pentru o alta aplicatie. Acest lucru ii confera portabilitate.

Cocncluzie:

Un Repository:

- Nu este un layer de acces de date
- Ofera un nivel inalt de manipulare a datelor
- Este o colectie de cai aggregate
- Ofera un mecanism usor de a manageui entitatile

Asadar, prin Domain-Driven Design: Repository, ce de regula este implementat pe baza unor patternuri, se intelege un serviciu ce cu ajutorul unei interfete globale poate oferi acces tuturor entitatilor si obiectelor valoare ce se afla intr-o colectie agregata.

Testarea unui repository se face prin testarea interfetelor ce il defines.

https://airbrake.io/blog/software-design/domain-driven-design

https://pehapkari.cz/blog/2018/02/28/domain-driven-design-repository/

https://blog.fedecarg.com/2009/03/15/domain-driven-design-the-repository/

https://lostechies.com/jimmybogard/2009/09/03/ddd-repository-implementation-patterns/