Iriza Andrea-Cassiana

Anul 4, IS, gr. 3

Tema2 PSSC

DDD-repository

Repository-urile joacă un rol important în DDD, vorbesc limba domeniului și se comportă ca un mediator între domeniu și straturile de cartografiere a datelor. Oferă o limbă comună tuturor prin traducerea terminologiei tehice în terminologia afacerilor. Sunt create pentru agregate și lucrează cu întregul agregat, nu doar cu o parte din el.

Dacă ne-am imagina că un singur utilizator poate avea un sistem care rulează contiuu și are suficentă memorie, am putea avea toate obiectele în colecții de memorie care ne permit să stocăm, să primim și să eliminăm obiecte. Dar acest lucru nu este posibil pentru că, de obicei, se construiesc aplicații web de tipul request-process-response-die.

De accea folosim repository, ne prefacem că avem colecții inteligente care au responsabilitatea de a adăuga un obiect, de a obține obiecte prin identificare sau criterii mai complexe și de a elimina obiecte.

Repository-ul este implementat în stratul domeniu pentru că lucrează cu obiectele din domeniu, dar este doar o interfață pentru că stratul domeniu nu știe nimic despre bazele de date și nici despre spațiul de stocare.

Interfața generică

Interfața generică a repository-ului este ea însăși generică cu implementări pentru tipuri individuale de entități. Parametrul de tip generic denotă tipul entității repository-ului. O implementare corespunzătoare a interfeței generice poate fi proiectată astfel încât tot codul de acces la date care nu se schimbă de la un repository la altul poate avea un loc să “trăiască”. Un beneficiu este că cele mai moderne containere IoC permit configurarea containerului astfel încât cel mai derivat tip al repository-ului poate fi rezolvat.

Metode generice

În medoda generică a repository-ului, implementările nu expun metode pentru interogări specifice, nu este definit nicun repository pentru o entitate, dar sunt expuse metodele generale pentru interogări, persistență și recuperare. Avantajul este că nu este nevoie de crearea unei interfețe pentru fiecare entitate, în schimb, eliminăm responsabilitatea de a forma întrebări specifice apelanților. Dezavantajul este că pierdem metode de interogare. Cu un repository de metode generice există o singură implementare. Crearea unui ICustomerRepository derivat dintr-un repository de metode generice ar fi destul de ciudat.

Rolul încapsulat specific repository-ului

O clasă care face o interogare împotriva clienților nu va merge apoi să îi șteargă. Dar este posibil să interogăm alte scopuri, cum ar fi afișarea informațiilor despre reguli de afaceri. În loc de un repository de tip catch-all care expune fiecare metoda, putem aplica Pricipiul Segregation Interface și definim interfețe care expun doar ceea ce are nevoie o singură clasă. O implementare cu un singur repository implementează toate interfețele repository-ului produsului și doar singura metodă necesară este expusă și folosită de apelant.

Responsabilitatea persistenței

Repository-ul poate fi responsabil pentru obiectele persistente. Ar fi logic să avem o metoda de salvare care să reziste instantaneu uui obiect, dar nu există un astfel de caz de utilizare cu colecțiile de memorie. Soluția este că persistența obiectelor nu este responsabilitatea repository-ului, altcineva este responsabil.

Implementarea concretă

Putem stoca obiecte din domeniu într-o baza de date relațională, într-o bază de date a documentelor, într-un sistem extern conectat prin API. Toate acestea reprezintă infrastructura domeniului, deci implementarea repository-ului se află în stratul de infrastructură.

Inversiunea dependenței

Într-un model activ de înregistrare a persistenței și modele similare, domeniul depinde de infrastructură, dar am creat o interfață repository de domenii, iar implemtarea este infrastructura. Stratul de domeniu este încă independent.

Implementarea memoriei

Implementarea memoriei este cel mai simplu de făcut. O implementare care păstrează obiectele în timpul unei vieții și nu le persistă deloc. Este utilă pentru teste complexe sau pentru module. Integrăm mai multe părți ale sistemului, dar nu folosim implementarea reală a repository. Deoarece totul este în memorie, testele sunt rapide și încă testează întreaga componentă sau modul.

Testul interfeței

Ideea de bază este că testul așteaptă implementarea interfeței. Responsabilitatea testului nu este de a crea obiectul testat, testul este o clasă abstractă, iar testul de implementare extinde testul abstract.

TL, DR

Repository-urile sunt colecții persistente care ne permit să presupunem că sistemul este în memorie. Funcționează cu agregate complete și este o interfață în stratul domeniu. Implementarea concretă se face prin infrastructura folosită.

În concluzie, un repository nu este un strat de acces la date, oferă un nivel mai idicat de manipulare a datelor, este ignoranța persistenței, este o colocție de rădăcini de agregate, oferă un mecanism de gestionare a entităților.