Documentazione Gruppo 3

Three Layer Architecture

Descrizione

L'architettura scelta è un'architettura a tre livelli composta da:

- Presentation Layer: consente all'utente di interagire con l'applicazione. Questo layer presenta i dati all'utente e ne permette la manipolazione sfruttando i servizi offerti dal business logic layer.
- Business Logic Layer: consente di applicare la logica necessaria sui dati, al contempo offrendo servizi di utilità al presentation layer inerenti all'elaborazione dei dati ed utilizza i servizi offerti dal data access layer per recuperare e immagazzinare i dati in un persistent storage. Tale layer non è progettato per essere dipendente da un client specifico, ma può essere utilizzato da più applicazioni favorendone il riutilizzo. Il meccanismo di storage, implementato nel data access layer, è trasparente a questo livello dato l'utilizzo di interfacce per la manipolazione dei dati mediante le operazioni CRUD. Di conseguenza, risulta semplificata un'eventuale transizione ad un differente metodo di memorizzazione dei dati.
- Data Access Layer: è composto dai componenti per l'accesso ai dati ed interagisce direttamente con lo specifico persistent storage. In particolare, è stato impiegato un database ospitato su un server remoto.

I dati tra i tre livelli vengono trasferiti sottoforma di *data transfer objects.* Tali oggetti non implementano alcuna logica; il loro scopo è solo quello di mantenere informazioni legate ad una singola entità del problema di interesse.

Benefici

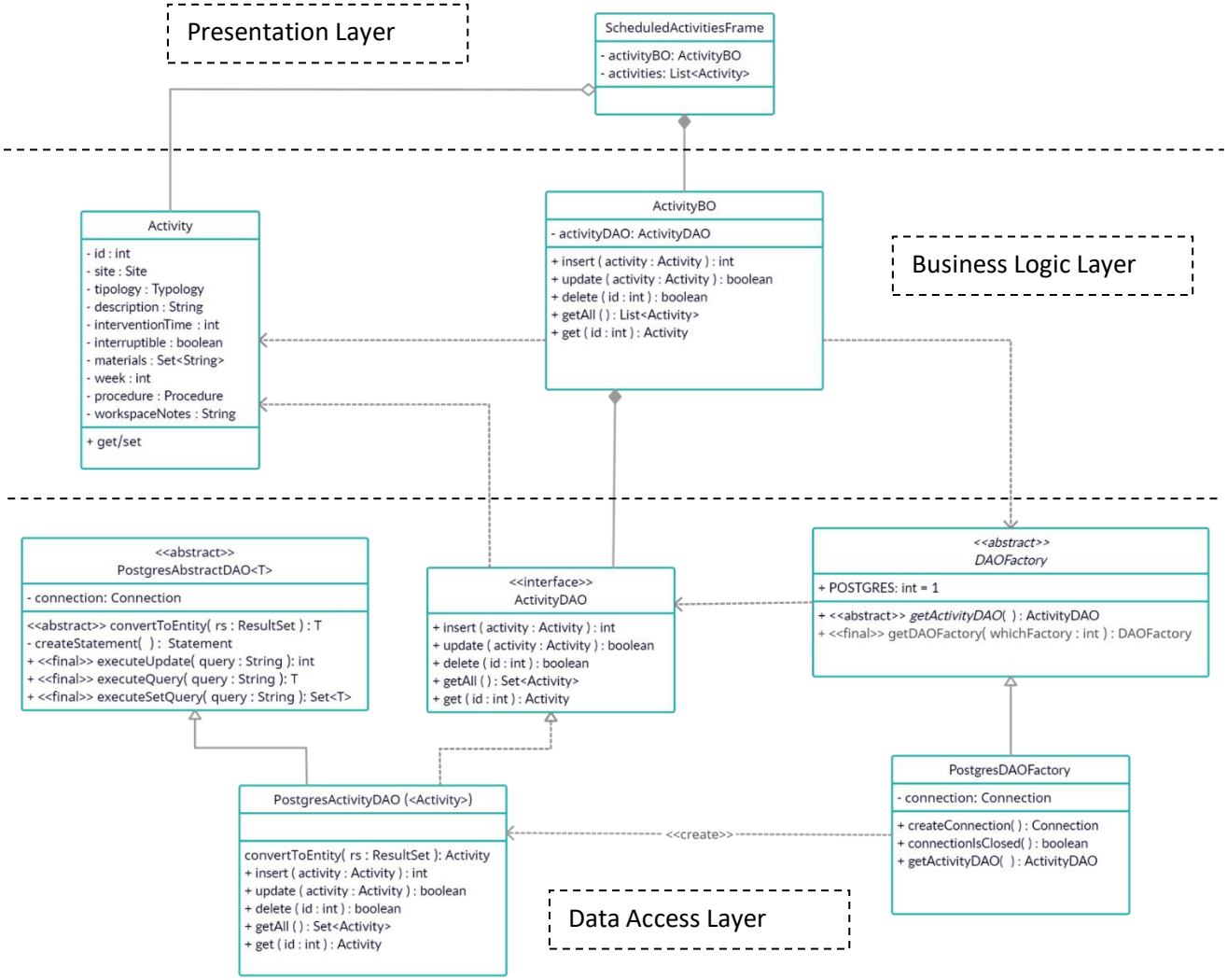
L'architettura fornisce la possibilità di apportare modifiche ad uno strato senza impattare sugli altri strati a patto che le interfacce definite per l'interazione tra livelli adiacenti rimangano immutate. Ad esempio, un cambiamento del persistent storage richiederebbe la sola implementazione delle classi appropriate nel data access layer, mentre il business logic layer (come anche il presentation layer) non sarebbe influenzato da tale modifica. Tale architettura consente uno sviluppo parallelo dei diversi strati una volta definite le interfacce fra layer adiacenti. Una scelta di questo tipo, inoltre, favorisce la security dato che il client ha accesso ai dati solamente tramite il business logic layer.

Svantaggi

L'architettura presenta una maggiore complessità nella comunicazione fra i vari livelli rispetto ad altre architetture con un numero inferiore di livelli o basate su differenti modelli architetturali. Inoltre, tale architettura riduce le performance del sistema dato che, siccome un layer può comunicare solamente con i layer adiacenti, anche un'informazione che non deve subire alcuna manipolazione logica deve comunque attraversare il business logic layer prima di essere presa in carico dal data access layer o dal presentation layer. Per un motivo analogo, l'aggiunta di una nuova funzionalità nel presentation layer richiede la predisposizione delle operazioni necessarie in tutti i layer inferiori.

Diagramma UML

Di seguito è riportato un frammento del diagramma UML per dimostrare le principali interazioni fra le classi dei diversi layer per la visualizzazione di una lista di attività. In tale frammento si può notare l'utilizzo della classe `Activity` per rappresentare i relativi transfer object utilizzati per lo scambio dei dati fra i vari livelli.
Inoltre, è stato utilizzato il design pattern Abstract Factory per la creazione dei data access object specifici di una certa famiglia così da prevedere una più facile migrazione verso altri meccanismi di storage.


Presentation Layer
ScheduledActivitiesFrame
- activityBO: ActivityBO
- activities: List<Activity>
Activity
- id : int
- site : Site
- tipology : Typology
- description : String
- interventionTime : int
- interruptible : boolean
- materials : Set<String>
- week : int
- procedure : Procedure
- workspaceNotes : String
+ get/set
ActivityBO
- activityDAO: ActivityDAO
+ insert (activity : Activity) : int
+ update (activity : Activity) : boolean
+ delete (id : int) : boolean
+ getAll ( ) : List<Activity>
+ get (id : int) : Activity
Business Logic Layer
<<abstract>>
PostgresAbstractDAO<T>
- connection: Connection
<<abstract>> convertToEntity(rs : ResultSet) : T
- createStatement( ) : Statement
+ <<final>> executeUpdate(query : String): int
+ <<final>> executeQuery(query : String): T
+ <<final>> executeSetQuery(query : String): Set<T>
<<interface>>
ActivityDAO
+ insert (activity : Activity) : int
+ update (activity : Activity) : boolean
+ delete (id : int) : boolean
+ getAll ( ) : Set<Activity>
+ get (id : int) : Activity
<<abstract>>
DAOFactory
+ POSTGRES: int = 1
+ <<abstract>> getActivityDAO( ) : ActivityDAO
+ <<final>> getDAOFactory(whichFactory : int) : DAOFactory
PostgresActivityDAO (<Activity>)
convertToEntity(rs : ResultSet): Activity
+ insert (activity : Activity) : int
+ update (activity : Activity) : boolean
+ delete (id : int) : boolean
+ getAll ( ) : Set<Activity>
+ get (id : int) : Activity
<<create>>
PostgresDAOFactory
- connection: Connection
+ createConnection( ) : Connection
+ connectionIsClosed( ) : boolean
+ getActivityDAO( ) : ActivityDAO
Data Access Layer