Università degli Studi di Salerno
Corso di Ingegneria del Software

LetsMeet
ODD – Object Design Document
Versione 1.2

Docente:
Andrea De Lucia

Studenti:
Vittorio Aiello
Gerardo Benevento
Raffaele Sansone

Data: 24/03/2019

Progetto: LetsMeet Documento: ODD	Versione: 1.2 Data: 24/03/2019

Coordinatore del progetto:

Nome	**Matricola**
Vittorio Aiello	0512104524

Partecipanti:

Nome	**Matricola**
Vittorio Aiello	0512104524
Gerardo Benevento	0512104584
Raffaele Sansone	0512104974

Scritto da:	Gerardo Benevento, Raffaele Sansone, Vittorio Aiello

Revision History

Data	Versione	Descrizione	Autore
21/12/2018	**1.0**	**Prima stesura del ODD**	**GB; VA; RS**
3/02/2019	**1.1**	**Revisione del ODD**	**GB; VA; RS**
24/03/2019	**1.2**	**Revisione ODD**	**GB; VA; RS**

Sommario

1. Introduzione

Dopo aver stilato il documento di Requirements Analysis (RAD) e il documento di System Design(SDD) in cui vi è una descrizione sommaria di ciò che sarà il nostro sistema, definendo i nostri obiettivi ma tralasciando gli aspetti implementativi, andiamo ora a stilare il documento di Object Design che ha come obiettivo quello di produrre un modello che sia in grado di integrare in modo coerente e preciso tutte le funzionalità individuate nelle fasi precedenti.
In particolar modo, in tale documento si definiscono le interfacce delle classi, le operazioni, i tipi, gli argomenti e la signature dei sottosistemi definiti nel System Design. Inoltre, sono specificati i trade-off e le linee guida.

Comprensibilità vs Tempo:

Il codice del sistema deve essere comprensibile il più possibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Per rispettare queste linee guida il codice sarà accompagnato da commenti volti a semplificarne la comprensione.
Ovviamente questo comporterà un aumento del tempo di sviluppo del nostro progetto.

Prestazioni vs Costi:

Dato che il progetto ha un budget mensile di 5 euro in totale, per poter mantenere prestazioni elevate, in determinate funzionalità verranno utilizzati dei template open source esterni.
Per questo bisognerà limitare l'utilizzo di memoria RAM che di storage, per le restrizioni imposte da un sistema a basso costo
Bisognerà assicurarsi che il sistema garantisca prestazioni soddisfacenti, ovvero che soddisfino i requisiti non funzionali, anche se deployato su hardware non molto prestante

Interfaccia vs Usabilità:

L'interfaccia grafica è stata realizzata in maniera molto semplice, chiara e concisa, vengono utilizzati i form e pulsanti con lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale.

Sicurezza vs Efficienza:

Non si evincono trade-off tra sicurezza ed efficienza dall'analisi dei requisiti.
Le misure di sicurezza che verranno implementate sono presenti nel RAD

1.1 Linee Guida per la Documentazione delle Interfacce

Gli sviluppatori dovranno seguire determinate linee guida per la stesura del codice:

Naming Convention:

È buona norma utilizzare nomi:

- Descrittivi
- Pronunciabili
- Di uso comune
- Di lunghezza medio-corta
- Non abbreviati
- Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)
- Bisogna utilizzare la convenzione del CamelCase

Variabili:

● I nomi delle variabili devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola (camelCase). In ogni riga vi deve essere una sola dichiarazione di variabile e va effettuato l'allineamento per migliorare la leggibilità.
Esempio: idProdotto, nomeProdotto

● In determinati casi, è possibile utilizzare il carattere underscore "_", ad esempio quando si fa uso di variabili costanti oppure quando si fa uso di proprietà statiche.
Esempio: AGGIUNGI_PRODOTTO

Metodi:

● I nomi dei metodi devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. Di solito il nome del metodo è costituito da un verbo che identifica un'azione, seguito dal nome di un oggetto.
I nomi dei metodi per l'accesso e la modifica delle variabili dovranno essere del tipo getNomeVariabile() e setNomeVariabile(). Se viene dichiarata una variabile all'interno di un metodo quest'ultima deve essere dichiarata appena prima del suo utilizzo e deve servire per un solo scopo, per facilitare la leggibilità.
Esempio: getId(), setId()

● I metodi vanno corredati di documentazione javadoc
La descrizione del metodo deve includere anche informazioni riguardanti gli argomenti, il valore di ritorno, le eccezioni. I metodi devono essere raggruppati in base alla loro funzionalità.

Classi, Servelt e pagine:

● I nomi delle classi e delle pagine devono iniziare con la lettera maiuscola, e anche le parole successive all'interno del nome devono iniziare con la lettera maiuscola.
I nomi delle classi e delle pagine devono essere evocativi, in modo da fornire informazioni sullo scopo di quest'ultime.
Esempio: GestioneFornitori.java
Ogni file sorgente .java contiene una singola classe principale e dev'essere strutturato in un determinato modo:

● Una breve introduzione alla classe
L'introduzione indica: l'autore, la versione e la data.

```
/**
* sommario dello scopo della classe.
*
* @author [nome dell'autore]
* @version [numero di versione della classe]
*/
```

● L'istruzione include che permette di importare all'interno della classe gli altri oggetti che la classe utilizza.
● La dichiarazione di una classe è caratterizzata da:
● Dichiarazione della classe pubblica
● Dichiarazioni di costanti
● Dichiarazioni di variabili di classe
● Dichiarazioni di variabili d'istanza
● Costruttore
● Commento e dichiarazione metodi e variabili

Si dovranno seguire le convezioni imposte dai design pattern di scelta, seguendo anche le comuni convenzioni del linguaggio Java.

ESEMPIO:

```
class Prodotto {
/**
* @var int id del prodotto
*/
private id;
/**
* @var string nome il nome del prodotto
*/
private nome;
/**
* Costruttore di Prodotto
* @param string id L'identificativo del prodotto
* @param string nome Il nome del Prodotto
*/
public function _construct(id_2, nome_2) {
this.id = id_2;
this.nome = nome_2;
}
/**
* restituisce l'id del prodotto
* @return string L'id del Prodotto
*/
public function getId() {
return this.id;
}
/**
* restituisce il nome del prodotto
* @return string Il nome del Prodotto
*/
public function getNome() {
return this.nome;
}
/**
* Setta l'id del Prodotto
* @param string id L'id del prodotto
*/
public function setId(id_2) {
this.id = id_2;
}
/**
* Setta il nome del Prodotto
* @param string nome Il nome del Prodotto
*/
public function setNome(nome_2) {
this.nome = nome_2;
}
}
```

Il formato delle classi Servlet, invece, deve:

- Contenere un costruttore, anche se vuoto
- Contenere i metodi doGet() e/o doPost()
- Contenere il codice Javadoc per la classe e per i metodi in modo da definire lo scope della Servlet e le operazioni dei due metodi.

Un **esempio** più pratico del corretto formato lo vediamo nel frammento di codice seguente:

```java
package src;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 *  Descrizione di cosa fa la Servlet
 */

@WebServlet("/ExampleServlet")
public class ExampleServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;


    public ExampleServlet() {
    }

    /**
     * Descrizione di cosa fa il doGet
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        /* Content of GET */
    }

    /**
     * Descrizione di cosa fa il doPost
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

        /* Content of POST */
    }

}
```

È possibile aggiungere qualsiasi clausola si voglia all'interno del codice Javadoc della classe e dei metodi. Inoltre, è possibile anche implementare metodi o funzioni proprie della classe in modo da rendere le sue operazioni più modulari.

Il formato della classe riguardante la pagina sarà quello di un file di tipo .html o .jsp, quindi:

```
<html>

        <head>

        <head>

         <body>

                <div>

                        Contenuto DIV

                <div>

        <body>

<html>
```

Il codice HTML deve essere indentato in maniera tale da poter mantenere sulla stessa colonna il TAG di apertura e di chiusura. Inoltre, il contenuto di un TAG si distanzierà dalle clausole dell'elemento in cui è contenuto di una distanza pari ad 1 TAB.
I Tag che non hanno una clausola di chiusura seguiranno solamente la seconda condizione.

2.0 Design pattern globali

Singleton Design Pattern: Utilizzato per ottenere una sola instanza del DataSource

Data Access Object (DAO) Pattern: Utilizzato per ottenere un'interfaccia omogenea per le varie implementazioni dello storage layer

Abstract Factory Pattern: per semplificare la creazione degli oggetti DAO in quanto la loro inizializzazione potrebbe differire profondamente a seconda dell'implementazione

3.0 Package Component

La gestione del nostro sistema è suddivisa in tre livelli (three-tier):

- Interface layer
- Application Logic layer
- Storage layer

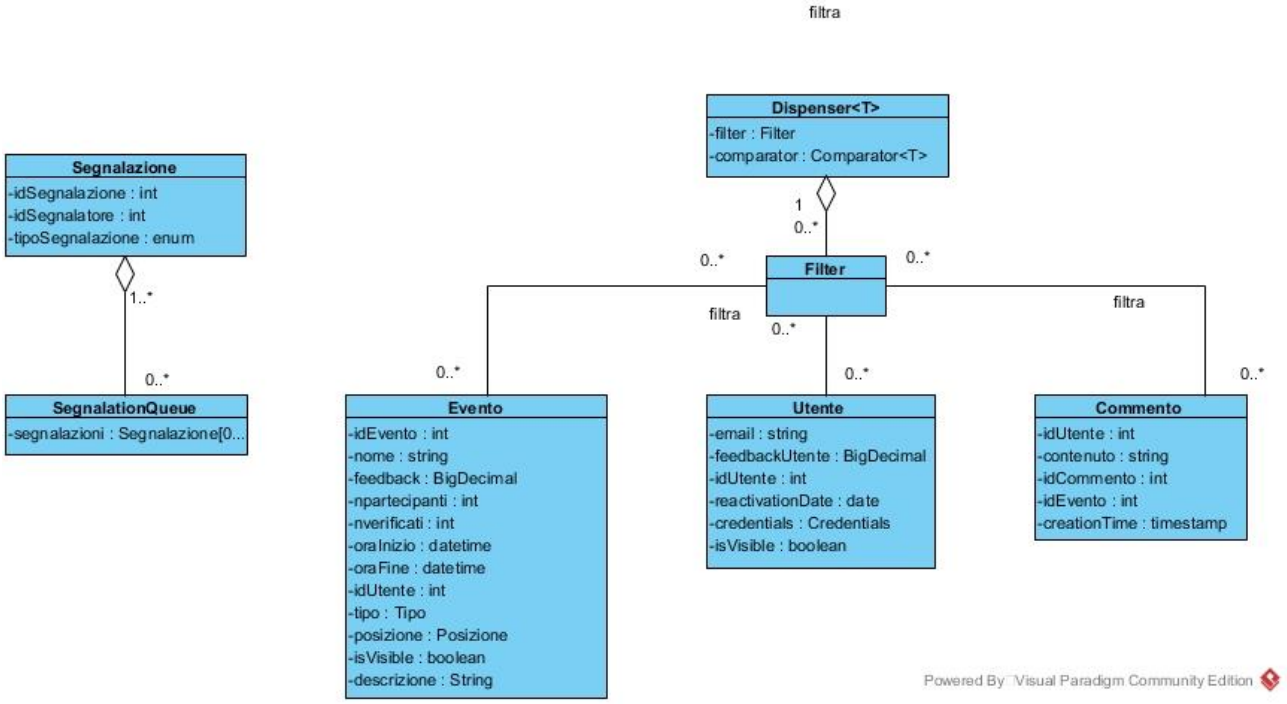
Il package "LetsMeet" contiene sottopackage che a loro volta inglobano classi atte alla gestione delle richieste utente. Le classi contenute nel package svolgono il ruolo di gestore logico del sistema.

Interface layer	Rappresenta l'interfaccia del sistema, ed offre la possibilità all'utente di interagire con quest'ultimo, offrendo sia la possibilità di inviare input, che di visualizzare dati.

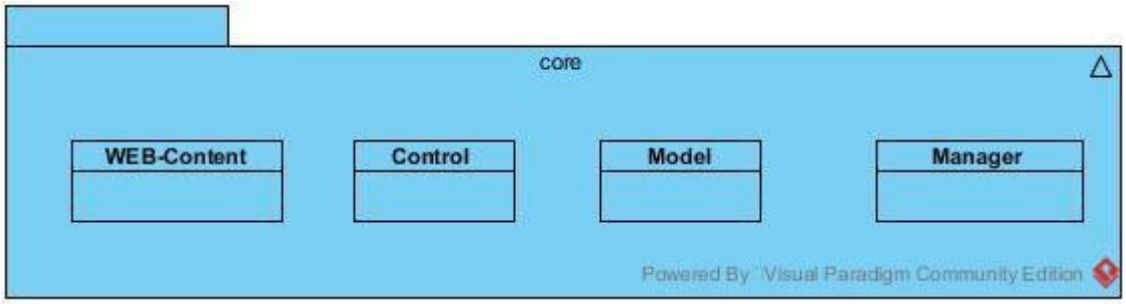
Application Logic layer	Ha il compito di elaborare i dati da inviare al client, e spesso grazie a delle richieste fatte al database, tramite lo Storage Layer, accede ai dati persistenti. Si occupa di varie gestioni quali: 1. Gestione Account 2. Gestione Eventi 3. Gestione Segnalazione

Storage layer	Ha il compito di memorizzare i dati persistenti del sistema, utilizzando un DBMS, inoltre riceve le varie richieste dall' Application Logic layer inoltrandole al DBMS e restituendo i dati richiesti.

Dopo una fase di object analysis sono stati individuati i seguenti oggetti del dominio delle soluzioni, oltre alle interfacce e classi del package manager che saranno descritte nella sezione 4.1



- WebContent: contiene tutte le risorse che vanno a costituire il sito web e implementa il layer di presentazione
- Control: contiene tutte le classi che permettono di gestire la logica di controllo.
- Model: contiene tutte le classi che permettono la manipolazione dei dati persistenti dell'applicazione.
- Manager: contiene tutte le classi che permettono la connessione al DBMS.



3.1 Package Model

Di seguito verrà fornita una suddivisione dei diversi Manager/Dao che verranno utilizzati all'interno del sistema e che risiederanno del package Model:

Classe	Descrizione
UtenteBean.java	Descrive l'utente registrazione

EventoBean.java	Descrive un evento
SegnalazioneBean.java	Descrive una segnalazione
CommentoBean.java	Descrive un commento
RatingBean.java	Descrive un rating di un evento
PartecipazioneBean.java	Descrive una partecipazione ad un evento ad evento
SegnalazioneEventoBean.java	Descrizione una segnalazione ad un evento svolto
SegnalazioneCommentoBean.java	Descrizione una segnalazione ad un commento
PosizioneBean.java	Descrizione una posizione ad un evento
TipoBean.java	Descrive il tipo dell'evento
CodiceRilasciatoBean.java	Descrive un codice rilasciato ad un utente durante la registrazione
CredentialsBean.java	Descrive le credenziali di un utente
SuperAdminBean.java	Descrive un super-admin del sistema
PosizioneBean.java	Descrive una posizione

3.2 Package WebContent

Il package view include tutte le pagine JSP, ossia quelle pagine che gestiscono la visualizzazione dei contenuti da parte di un utente del sistema. Contiene inoltre file js, css e immagini necessarie suddivisi rispettivamente nei sottopackage: js, css, images. Di seguito, ecco riportato il package che raggruppa tutte le pagine del sistema che hanno tale caratteristica:

Classe	**Descrizione**
homePage.jsp	Pagina principale della piattaforma, contenente una versione rimpicciolita di una mappa.
profilo.jsp	Pagina di profilo dell'utente con le sue info riguardanti feedback, nome, e-mail e gli eventi a cui ha partecipato e che ha creato.
infoEvento.jsp	Pagina che mostra le informazioni di un evento più i suoi commenti.
registrazione.jsp	Pagina di registrazione della piattaforma.
login.jsp	Pagina di login della piattaforma.

3.3 Package Control

Il package control include tutte le classi Servlet che rappresentano la logica applicativa della web platform. Il sistema vede la separazione di tali classi in sottopackage differenziati dai vari sottosistemi della web platform proposta. Di seguito presentiamo il contenuto di tale package con la seguente tabella:

Classe
Gestione Account
Gestione Evento
Gestione Segnalazione
Gestione Search

3.3.1 Package Gestione Account

Il package utente include tutte quelle classi Servlet adibite a svolgere una funzionalità del sottosistema di gestione dell'utente. Di seguito presentiamo il contenuto di tale package con la seguente tabella:

Classe	Descrizione
LoginControl	Permette all'utente di autentificarsi nel sistema
LogoutControl	Permette all'utente di scollegarsi al sistema
RegistrazioneControl	Permette all'utente di registrazione dell'utente nel sistema
ProfiloControl	Permette di poter visualizzare le informazioni di un profilo di un utente
RegistrazioneModeratoreControl	Permette di rendere un utente moderatore della piattaforma
ProfiloEsternoControl	Permette di poter visualizzare le informazioni di un profilo esterno al proprio
AuthFilter	Filtro che permette l'autenticazione alla piattaforma

3.3.2 Package Gestione Evento

Il package evento include tutte quelle classi Servlet adibite a svolgere una funzionalità del sottosistema di gestione dell'evento. Di seguito presentiamo il contenuto di tale package con la seguente tabella:

Classe	Descrizione
EventoControl	Permette di poter creare un evento sulla piattaforma
EventoInfoControl	Permette di poter visualizzare le informazioni di un evento
RatingControl	Permette di poter dare un voto all'evento
ValidationControl	Permette di poter validare la presenza ad un evento
CancellazioneEventoControl	Permette di poter cancellare un evento creato sulla piattaforma
PartecipazioneControl	Permette di poter indicare la partecipazione ad un evento
CommentoControl	Permette di scrivere un commento ad un evento

3.3.4 Package Gestione Segnalazione

Il package gestione segnalazione include tutte quelle classi Servlet adibite a svolgere una funzionalità del sottosistema di gestione delle segnalazioni. Di seguito presentiamo il contenuto di tale package con la seguente tabella:

Classe	Descrizione
SegnalazioneCommentoControl	Permette di creare una segnalazione per un commento
SegnalazioneEventoControl	Permette di creare una segnalazione per un evento
ModeratoreSegnalazioneControl	Permette di poter accettare, rifiutare oppure ignorare una segnalazione
SegnalationQueue	Permette di gestire una coda di segnalazioni in maniera concorrente

3.3.5 Package Search

Il package Search include tutte quelle classi Servlet adibite a svolgere una funzionalità del sottosistema di ricerca all'interno della piattaforma. Di seguito presentiamo il contenuto di tale package con la seguente tabella:

Classe	Descrizione
SearchUserControl	Permette di poter trovare utenti della piattaforma
SearchEventControl	Permette di avere informazioni generali sugli eventi ritornando una lista di eventi secondo particolari criteri
SearchCommentControl	Permette di avere i commenti relativi all'evento indicato
Filter	Permette di filtrare elementi tramite l'ereditarietà implementativa

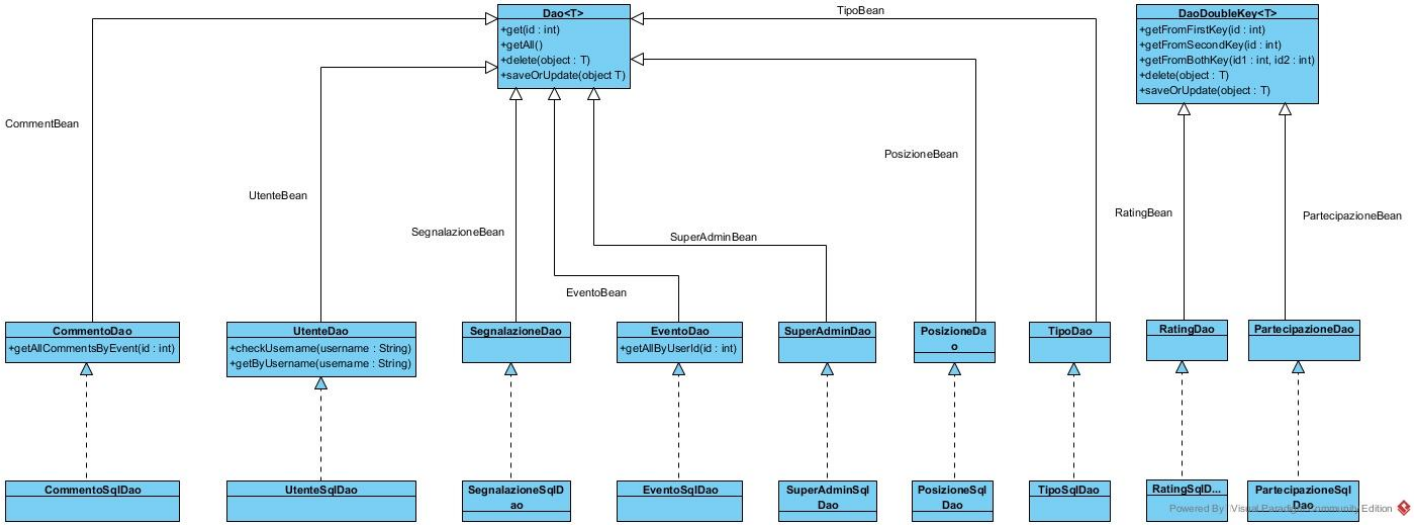
3.4 Package Manager

Classe	Descrizione
Dao	Interfaccia che permette l'interazione con dati
DaoDoubleKey	Interfaccia che permette l'interazione con dati con chiave composta
SqlDao	Implementazione di Dao utilizzando un database relazionale mysql
SqlDaoDoubleKey	Implementazione di DaoDoubleKey utilizzando un database relazionale mysql

4.0 Class Interface

4.1 Package:Manager

In questo package sono state identificate gli oggetti del dominio delle soluzioni indicati nel seguente class diagram



4.1.1 Interface Dao:

get(int id):

Pre: id >= 0
Post://
Invariante://

getAll():

Pre://
Post://
Invariante://

saveOrUpdate(object:T):

Pre: object != null
Post://
Invariante://

delete(object:T):

Pre: object != null

Post://
Invariante://

4.1.2 Interface DaoDoubleKey:

getAllForFirstKey(id:int):

Pre: id >= 0
Post://
Invariante://

getAllForSecondKey(id:int):

Pre: id >= 0
Post://
Invariante://

getAllBothKey(id1:int, id2:int):

Pre: id1 >= 0 AND id2 >= 0
Post://
Invariante://

saveOrUpdate(object:T):

Pre: object != null
Post://
Invariante://

delete(object:T):

Pre: object != null
Post://
Invariante://

4.1.3 Interface CommentoDao

Estende Dao

getAllCommentsFromEventKey(id:int):

Pre: id >= 0
Post://
Invariante://

4.1.4 Interface EventoDao

Estende Dao

getAllByUserId(id:int):

Pre: id >= 0
Post://
Invariante://

4.1.5 Interface PartecipazioneDao

Estende Dao

4.1.6 Interface PosizioneDao

Estende Dao

4.1.7 Interface RatingDao

Estende DaoDoubleKey

4.1.8 Interface SegnalazioneDao

Estende Dao

4.1.9 Interface SuperAdminDao

Estende Dao

4.1.10 Interface TipoDao

Estende Dao

4.1.11 Interface UtenteDao

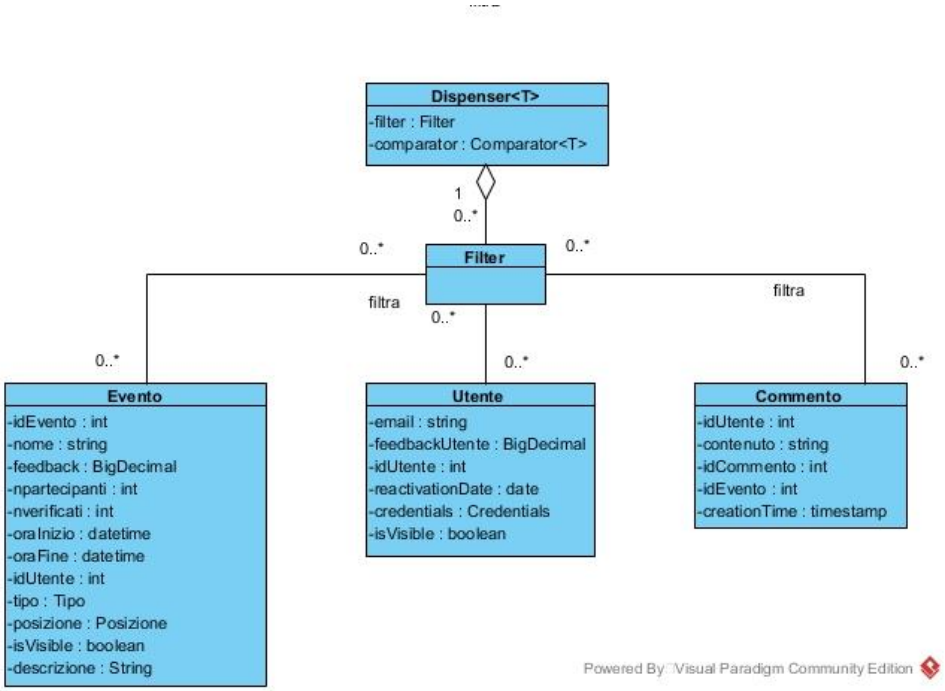
Estende Dao

checkUsername(username:String):
Pre: username != null
Post: //
Invariante: //

getByUsername(username:String):
Pre: username != null
Post: //
Invariante: //

4.2 Package filter

In questo package sono state identificate gli oggetti del dominio delle soluzioni indicati nel seguente class diagram



4.2.1 Interface Dispenser

dispense(filter:Filter, comparator:Comparator):

Pre: filter != null AND comparator != null

Post: //

Invariante: //

4.2.2 Interface Filter

check(item:T):

Pre: item != null

Post: //

Invariante: //

Nel package Control verranno analizzati tutti i sottopackage.
Queste andranno a costituire le API “LetsMeet”

Package: GestioneAccount

Context GestioneAccount:: LoginControl(username:string,password:string):
Pre: username !=null && password != null
Post://
Invarinate://

Context GestioneAccount:: LogoutControl():
Pre://
Post://
Invariante://

Context GestioneAccount:: RegistrazioneControl(username:string,password:string,email:string):
Pre: username !=null && password != null && email !=null
Post://
Invariante://

Cotext GestioneAccount:: RegistrazioneModeratoreControl(username:string):
Pre: username != null
Post: viene creato una nuova riga nella tabella Moderatore con riferimento all'utente con username = username passato.
Invariante: //

Context GestioneAccount:: ProfiloControl():
Pre: //
Post: //
Invariante: //

Context GestioneAccount:: ProfiloEsternoControl(idUtente:int):
Pre: idUtente esistente
Post: //
Invarinate : //

Package: GestioneEvento

Context GestioneEvento::
EventoControl(nome:string,oraInizio:date,oraFine:date,Posizione:posizione,tipo:tipo,descrizione:string):
Pre: nome != null && oraInizio >= currentDatetime && oraFine > orainizio && posizione != null && tipo != null
Post: viene creato l'evento indicato all'interno della tabella Evento
Invarinate: //

Context GestioneEvento:: EventoInfoControl(idEvento:int):
Pre: idEvento esistente
Post://
Invariante://

Context GestioneEvento:: RatingControl(idEvento:int,voto:boolean):
Pre: voto != null && idEvento esistente

Post: viene aggiunto un nuova riga nella tabella Rating e cambia il feedbackUtente ed feedbackEvento
Invariante: //

Context GestioneEvento:: ValidationControl(idEvento:int,longitudine,latitudine):
Pre: longitiudine != null && latitudine != null
Post: nverificati dell'evento viene incrementato di uno
Invariante: //

Context GestioneEvento:: PartecipationControl(idEvento:int):
Pre: idEvento esistente
Post: il numero dei partecipanti dell'evento aumentano
Invariante: //

Context GestioneEvento:: CancellazioneEventoControl(idEvento:int):
Pre: idEvento esistente
Post: l'Evento viene cancellato e non è più visibile
Invariante: //

Context GestioneEvento:: CommentControl(idEvento:int,contenuto:string):
Pre: idEvento esistente && contenuto <= 256 caratteri && contenuto != null
Post: //
Invariante: //

Package: GestioneSegnalazione
Context GestioneSegnalazione:: SegnalazioneCommentControl(idCommento:int):
Pre: idCommento esistente
Post: //
Invarinate: //

Context GestioneSegnalazione:: SegnalazioneEventoControl(idEvento:int):
Pre: idEvento esistente
Post: //
Invarinate: //

Context GestioneSegnalazione:: ModeratoreSegnalazioneControl(idSegnalazione:int, azione):
Pre: idSegnalazione esiste
Post: //
Invarinate: //

Package GestioneSearch

Context GestioneSearch:: SearchEventControl(filtro,comparatore,parametri:hashmap):
Pre: filtro e comparatore esistenti e parametri validi per quel filtro
Post: //
Invariante: //

Context GestioneEvento:: SearchUserControl(filtro,comparatore,parametri:hashmap):
Pre: filtro e comparatore esistenti e parametri validi per quel filtro

Post: //
Invariante: //

Context GestioneEvento:: SearchCommentControl(filtro,comparatore,parametri:hashmap):
Pre: filtro e comparatore esistenti e parametri validi per quel filtro
Post: //
Invariante: //