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 nostro progetto ha un budget mensile di 4 euro in totale, per poter mantenere prestazioni elevate, in determinate funzionalità verranno utilizzati dei template open source esterni.

Il budget mensile per il progetto è di ben 5 euro mensili in totale. Per questo bisognerà limitare l’utilizzo di memoria RAM che di storage, per le restrizioni imposte da un sistema a basso costo

Inoltre 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  
1.2 Linee Guida per la Documentazione delleInterfacceGli 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. 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.

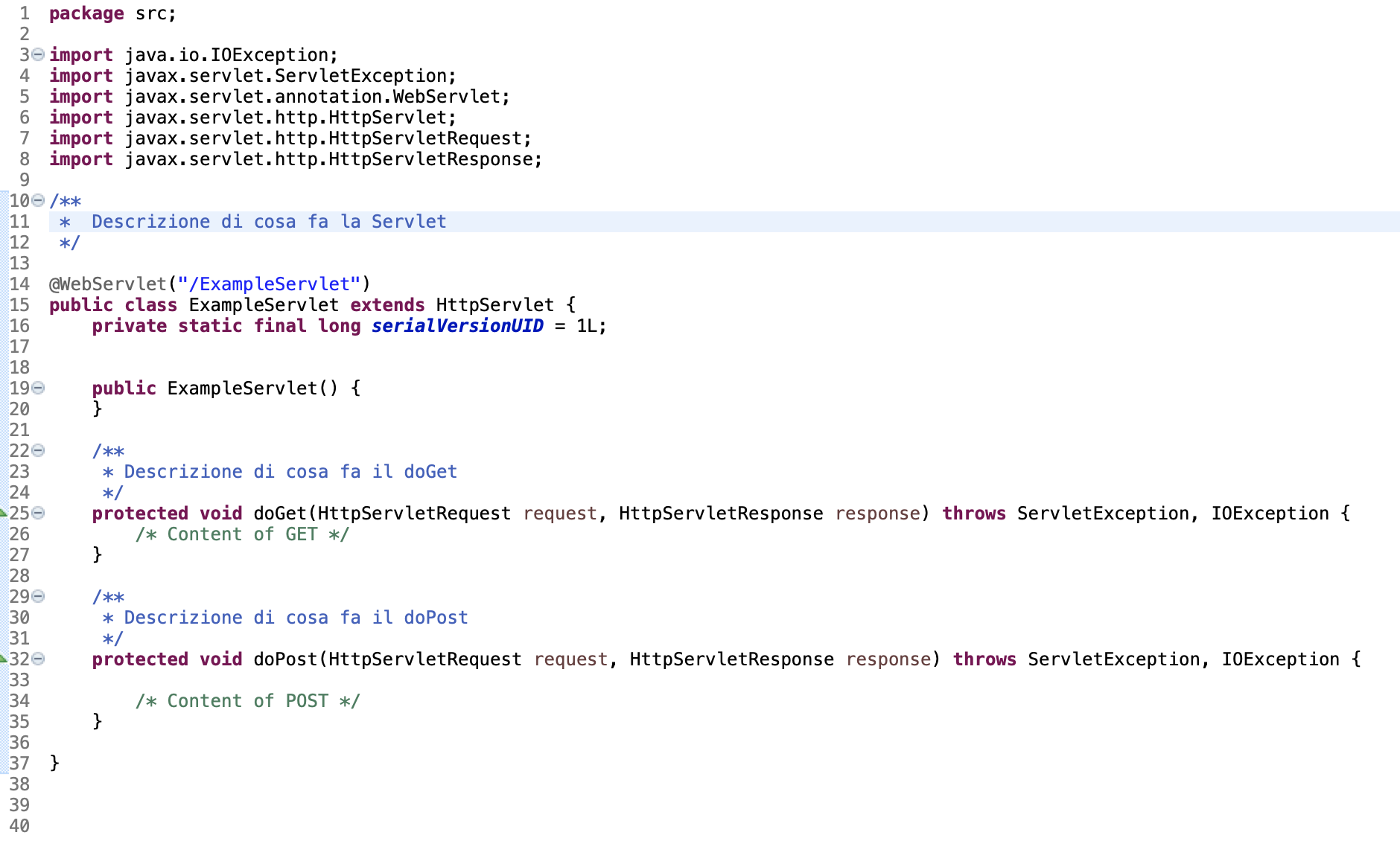
ESEMPIO:  
class Prodotto {  
/\*\*  
\* @var int id del prodotto  
\*/  
private id;  
/\*\*  
\* @var string nome il nome del prodotto  
\*/  
private nome;  
/\*\*  
\* Costruttore di Prodotto  
\* @param string id L’identificativo del prodotto  
\* @param string nome Il nome del Prodotto  
\*/  
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:



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.

**2.1 Design pattern globali**

**Command Design Pattern**: Letsmeet fa uso del Command Pattern che è uno dei pattern fondamentali, che permette di isolare la porzione di codice che effettua un'azione (eventualmente molto complessa) dal codice che ne richiede l'esecuzione; l'azione è incapsulata nell'oggetto Command. L'obiettivo è rendere variabile l'azione del client senza però conoscere i dettagli dell'operazione stessa.

**Singleton Design Pattern**: LetsMeet fa uso del Singleton Pattern nello Storage Layer per impedire la creazione di più di una istanza di oggetto di tipo Manager.

**Injection Pattern:** verrà utilizzato per permettere il testing.

**3.0 Package Component**

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

● Interface layer

● Application Logic layer

● Storage layer

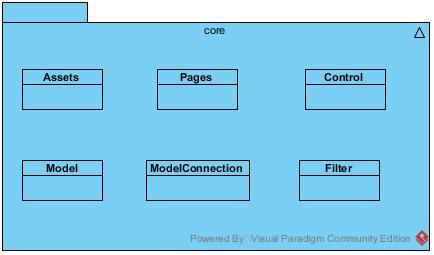
Il package “LetsMeet” contiene sottopackage che a loro volta inglobano classi atte alla gestione delle richieste utente. Le classi contenute nel package svolgono il ruolo di gestore logico del sistema.

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

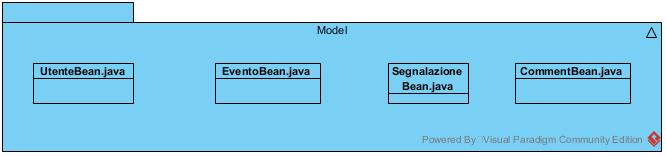
|  |  |
| --- | --- |
| Application Logic layer | Ha il compito di elaborare i dati da inviare al client, e spesso grazie a delle richieste fatte al database, tramite lo Storage Layer, accede ai dati persistenti.  Si occupa di varie gestioni quali:  1. Gestione Account  2. Gestione Eventi  3. Gestione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

* **Assets:** contiene tutti i file multimediali dell’applicazione.
* **Pages:** contiene le pagine dell’applicazione e ognuna di esse ha una cartella dove vi è definita la pagina (.html o .jsp), lo stile (.css) e l’eventuale script (.js).
* **Control:** contiene tutte le classi che permettono di gestire la logica di controllo dell’applicazione.
* **Model:** contiene tutte le classi che permettono la manipolazione dei dati persistenti dell’applicazione.
* **Model Connection:** contiene tutte le classi che permettono la connessione al DBMS.



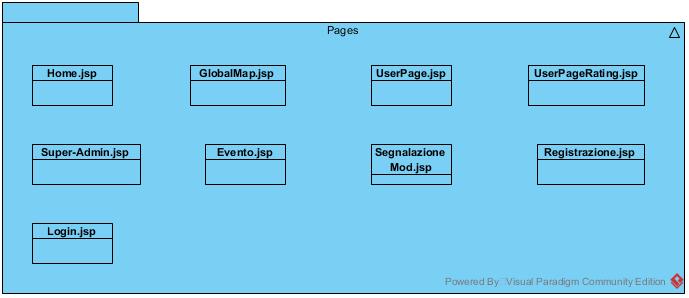
**3.1 Package Model**

****

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| UtenteBean.java | Descrive l’utente registrazione |
| EventoBean.java | Descrive un evento |
| SegnalazioneBean.java | Descrive una segnalazione |
| CommentoBean.java | Descrive un commento |
|  |  |
|  |  |

**3.2 Package Pages**

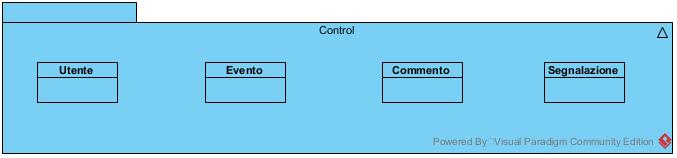
Il package view include tutte le pagine JSP, ossia quelle pagine che gestiscono la visualizzazione dei contenuti da parte di un utente del sistema. Di seguito, ecco riportato il package che raggruppa tutte le pagine del sistema che hanno tale caratteristica:



|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| Home.jsp | Pagina principale della piattaforma, contenente una versione rimpicciolita di una mappa. |
| GlobalMap.jsp | Pagina che contiene un ingrandimento della mappa |
| UserPage.jsp | Pagina di profilo dell’utente con le sue info riguardanti feedback, nome, e-mail e gli eventi a cui ha partecipato. |
| UserPageRating.jsp | Pagina che visualizza gli eventi a cui poter dare un voto. |
| Super-admin.jsp | Pagina del super-admin di controllo. |
| Evento.jsp | Pagina che mostra le informazioni di un evento più i suoi commenti. |
| SegnalazioneMod.jsp | Pagina che visualizza le segnalazioni fatte dagli utenti. |
| Registrazione.jsp | Pagina di registrazione della piattaforma. |
| Login.jsp | Pagina di login della piattaforma. |

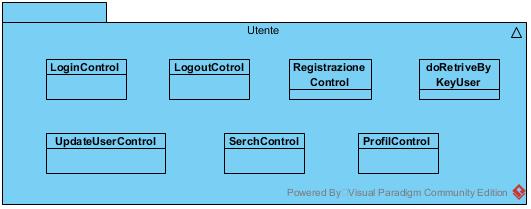
**3.3 Package Control**

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, illustriamo la suddivisione dei sottopackage con il seguente diagramma:

****

**3.3.1 Package Utente**

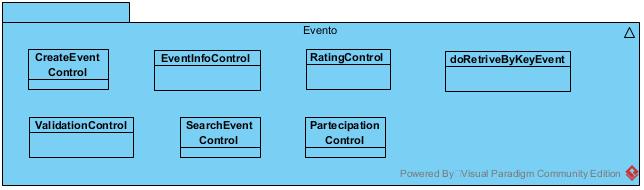
Il package utente include tutte quelle classi Servlet adibite a svolgere una funzionalità del sottosistema di gestione dell’utente. Di seguito presentiamo il contenuto di tale package con il seguente diagramma:

****

|  |  |
| --- | --- |
| **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 |
| SearchControl | Permette di poter ricercare un item nel sistema compresi utenti ed utenti |
| UpdateUserControl | Permette di rendere un utente moderatore della piattaforma |
| doRetriveByKeyUser | Permette di cercare un utente attraverso il proprio id univoco |

**3.3.2 Package Evento**

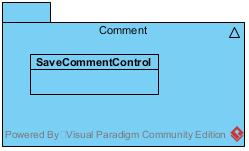
Il package evento include tutte quelle classi Servlet adibite a svolgere una funzionalità del sottosistema di gestione dell’evento. Di seguito presentiamo il contenuto di tale package con il seguente diagramma:

****

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| CreateEventControl | Permette di poter creare un evento sulla piattaforma |
| EventInfoControl | 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 |
| SearchEventControl | Permette la ricerca di eventi sulla piattaforma |
| PartecipationControl | Permette di poter indicare la partecipazione ad un evento |
| doRetiveByKeyEvent | Permette di poter trovare un evento mediante il suo id univoco |

**3.3.3 Package Commento**

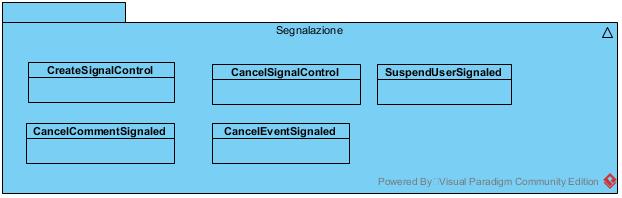
Il package commento include tutte quelle classi Servlet adibite a svolgere una funzionalità del sottosistema di gestione dei commenti. Di seguito presentiamo il contenuto di tale package con il seguente diagramma:

****

|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| SaveCommenstControl | Permette di salvare un commento scritto |

**3.3.4 Package Segnalazione**

Il package 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 il seguente diagramma:



|  |  |
| --- | --- |
| **Classe** | **Descrizione** |
| CreateSignalControl | Permette di creare una segnalazione |
| CancelSignalControl | Permette di poter cancellare una segnalazione |
| SuspendUserSignaled | Permette di poter sospendere un utente segnalato |
| CancelCommentSignaled | Permette di poter cancellare un commento segnalato |
| CancelEventSignaled | Permette di poter cancellare un evento segnalato |

**3.5 Package ModelConnection**

Il package contiene le classi per la connessione al db da parte del server

**4.0 Class Interface**

Nel package Control verranno analizzati tutti i sottopackege.

**Package: Utente**

Context Utente::searchControl(username):

pre: username!=null

Context Utente:: loginControl(username,password):

Pre: username!=null && password!= null

Context Utente:: logoutControl():

Pre:

Context Utente:: registrazioneUtente(username,password,email)

Pre: username!=null && password!= null && email!=null

Cotext Utente:: UpdateUserControl(username,email)

Pre:: username != null && email != null

Context Utente:: doRetiveByKey(idUtente)

Pre:: idUtente != null

**Package: Evento**

Context Evento:: CreateEventControl(nome,data,oraInizio,oraFine,creatore,posizione,tipo)

Pre:: nome != null && data != null && oraInizio != null && oraFine != null && creatore != null && posizione != null && tipo != null

Context Evento:: EventInfoControl()

Pre::

Context RatingControl(voto)

Pre:: voto!=null

Post:: evento.feedback ++ || evento.feedback –

Context ValidationControl(utente)

Pre:: utente is in evento.parteipanti

Post:: nverificati++

Context SearchEventControl(nome,posizione)

Pre:: nome != null || posizione != null

Context PartecipationControl(utente)

Post:: npartecipanti++

Context doRetiveByKey(idEvento)

Pre:: idEvento!= null

**Package:Commento**

Context SaveCommentControl(comment)

Pre:: comment != null

**Package: Segnalazione**

Context CreateSignalControl(utente)

Pre:: utente != null

Context CancelSignalControl(comment)

Pre:: comment != null

Context SuspendeSignalControl(signal)

Pre:: signal != null

Post:: signal.creatore.reactivationDate= date

Context CancelCommentSignaled(signal)

Pre:: signal.commentoSegnalato != null

Post:: signal = null

Context CancelEventSignaled(signal)

Pre:: signal.eventoSegnalato != null

Post:: signal = null