|  |
| --- |
| **logo_standard Università degli Studi di Salerno Corso di Ingegneria del Software** |

****

**WhereDoIEat  
Object Design Document  
Versione 1.0**

[**1. Introduzione**](file:///C:\Users\tanucc\Documents\tanucc\università\anno%20III\IS\progetto\ODD%20by%20gilby.docx#_tqboc5hf8lh8) **4**

1.1 Object Design Trade-Off 4

1.2 Linee guida per la Documentazione delle Interfacce 5

1.3 Definizioni,Acronimi e Abbreviazioni 6

1.4 Riferimenti 6

[**2.**](file:///C:\Users\tanucc\Documents\tanucc\università\anno%20III\IS\progetto\ODD%20by%20gilby.docx#_t5j8pddtg2l2) **Packages 7**

[2.1](file:///C:\Users\tanucc\Documents\tanucc\università\anno%20III\IS\progetto\ODD%20by%20gilby.docx#_c392ex7wtyt) WhereDoIEat 9

[2.2](file:///C:\Users\tanucc\Documents\tanucc\università\anno%20III\IS\progetto\ODD%20by%20gilby.docx#_ye1pg675k47i) Descrizione delle Classi 9

2.2.1 Model 9

2.2.2 Control 10

2.2.3 View 13

[**3. I**](file:///C:\Users\tanucc\Documents\tanucc\università\anno%20III\IS\progetto\ODD%20by%20gilby.docx#_ouyp94swbe30)**nterfacce delle Classi 17**

[**4.**](file:///C:\Users\tanucc\Documents\tanucc\università\anno%20III\IS\progetto\ODD%20by%20gilby.docx#_6tmevnpmt5ii) **Design Pattern 3**5

# **Introduzione**

## **Object design trade-offs**

Dopo aver stilato il documento di Requirements Analysis e il documento di System Design 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 è sprovvisto di budget, per poter mantenere prestazioni elevate, in determinate funzionalità verranno utilizzati dei template open source esterni, in particolare Bootstrap.

**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:**

La sicurezza, come descritto nei requisiti non funzionali del Requirements Analysis,

rappresenta uno degli aspetti importanti del sistema. Tuttavia, dati i tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su username e password degli utenti.

## **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
* Evitando la notazione ungherese
* Utilizzando solo caratteri consentiti (a-z, A-Z, 0-9)

**Variabili:**

* I nomi delle variabili devono iniziare con la lettera minuscola, e le parole successive con la lettera maiuscola. La dichiarazione delle variabili deve essere effettuata ad inizio blocco; in ogni riga vi deve essere una sola dichiarazione di variabile e va effettuato l’allineamento per migliorare la leggibilità.
* 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.

**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()
* Ai metodi va aggiunta una descrizione, la quale deve essere posizionata prima della dichiarazione del metodo, e deve descriverne lo scopo. 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 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. Ogni file sorgente .java contiene una singola classe 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]

\* @since [versione di partenza]

\*/

* L’istruzione import che permette di importare all’interno della classe gli altri oggetti che la classe utilizza.
* La dichiarazione di una classe è caratterizzata da:

1. Dichiarazione della classe pubblica
2. Dichiarazioni di costanti
3. Dichiarazioni di variabili di classe
4. Dichiarazioni di variabili d’istanza
5. Costruttore
6. Commento e dichiarazione metodi e variabili

## **Definizioni, acronimi e abbreviazioni**

**Acronimi:**

* RAD: Requirements Analysis Document
* SDD: System Design Document
* ODD: Object Design Document

**Abbreviazioni:**

* DB: DataBase

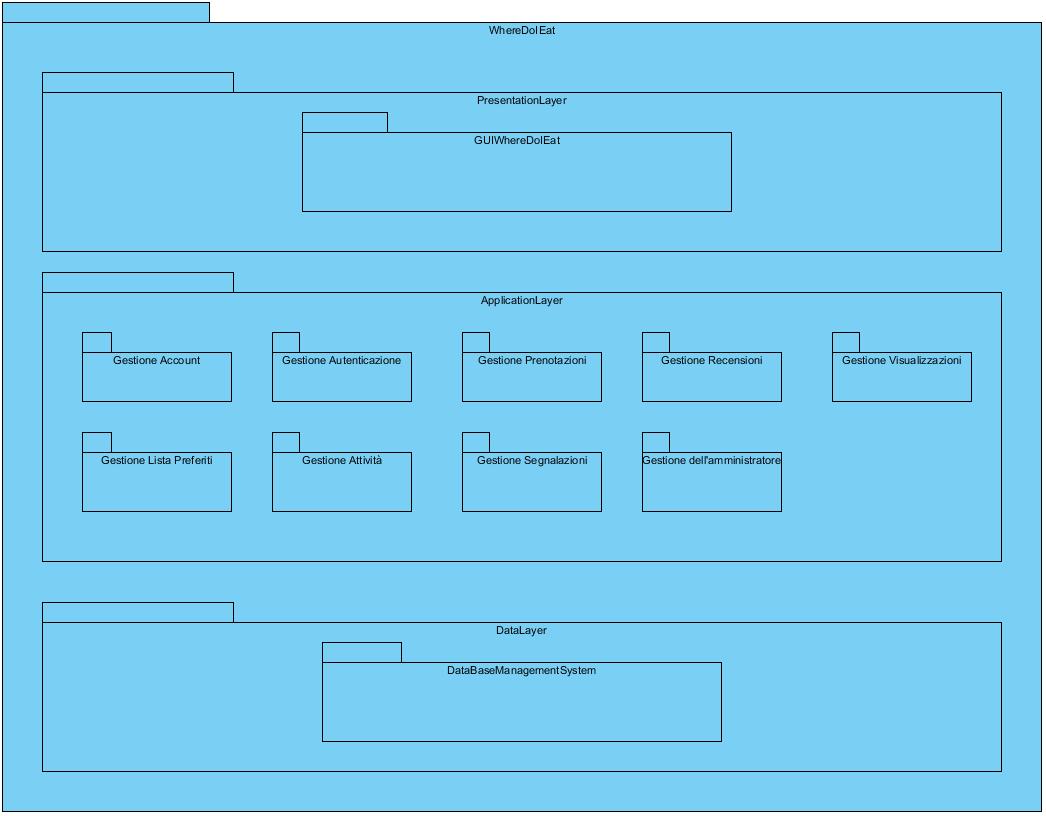
## **Riferimenti**

* B. Bruegge, A. H. Dutoit, Object Oriented Software Engineering - Using UML, Pattern and Java, Prentice Hall, 3rd edition, 2009
* Documento SDD del progetto WhereDoIEat
* Documento RAD del progetto WhereDoIEat

# **Packages**

La struttura del sistema WhereDoIEat è strutturata secondo una divisione in package e sottopackage che raggruppano le classi che hanno il compito di gestirne la logica in base alle richieste dell’utente che ne fa uso.

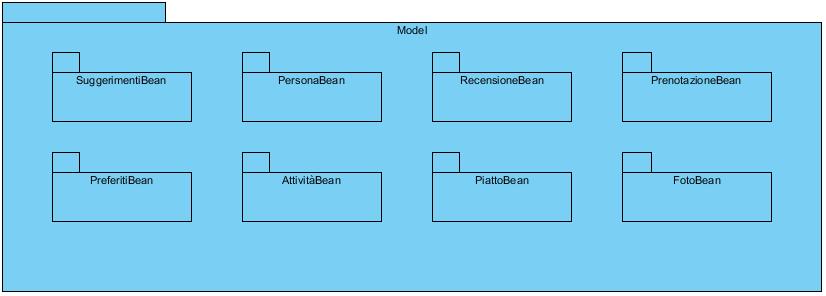
**2.1 WhereDoIEat**

****

**2.2 Descrizione delle classi**

**2.1.1 Model**

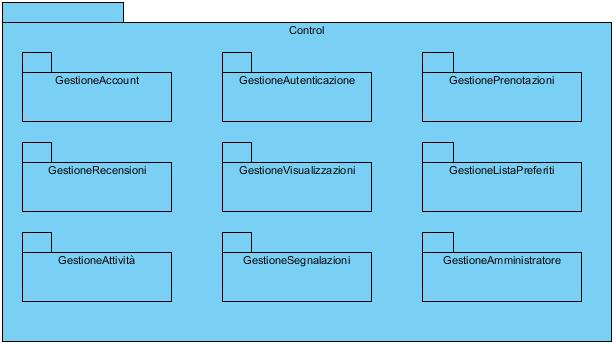
Il sottopackage “Model” è presentato nel seguente schema e contiene le classi java rappresentanti le entità presenti all’interno del sistema.

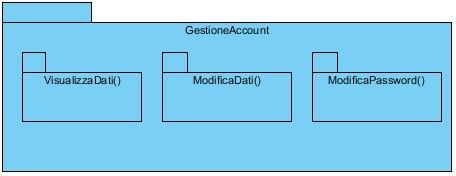


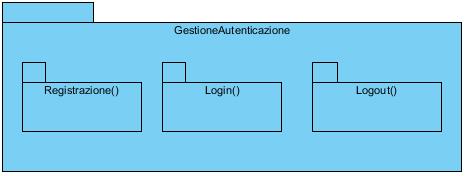
|  |  |
| --- | --- |
| **Classe:** | **Descrizione:** |
| SuggerimentiBean | Descrive un Suggerimento effettuato sul sistema. |
| PersonaBean | Descrive un utente. |
| FotoBean | Descrive una foto. |
| RecensioneBean | Descrive una recensione. |
| PrenotazioneBean | Descrive le prenotazioni. |
| PreferitiBean | Descrive le attività aggiunte alla lista preferiti. |
| AttivitàBean | Descrive un’attività. |
| PiattoBean | Descrive un piatto di un’attività. |

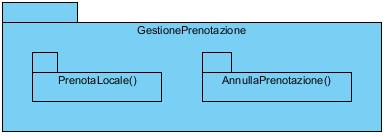
**2.2.2 Control**

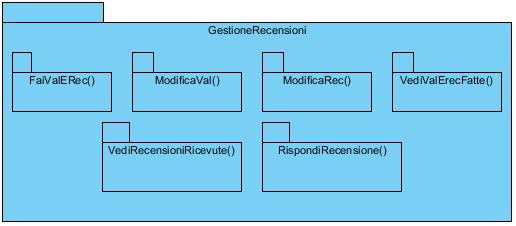
Il sottopackage “Control” è presentato nel seguente schema e contiene le classi Java che si occupano della logica di controllo del sistema.

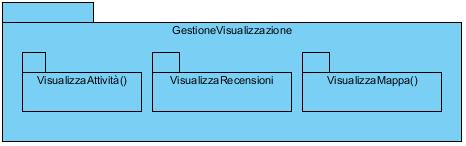


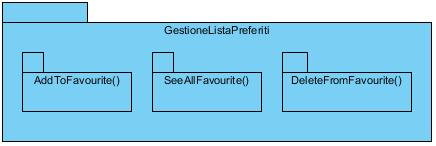


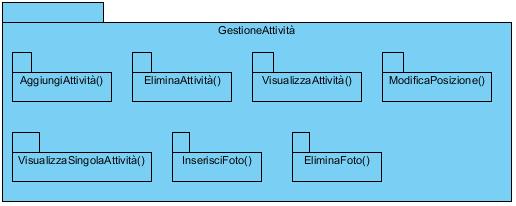


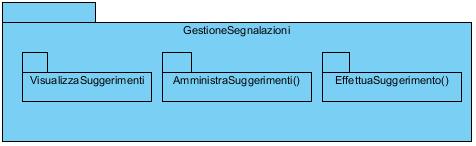


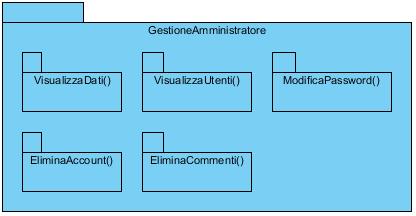






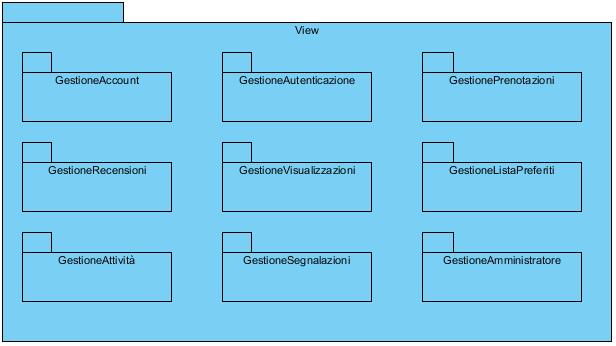


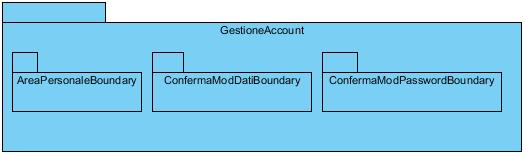


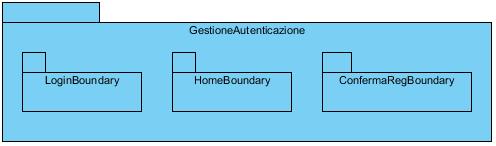


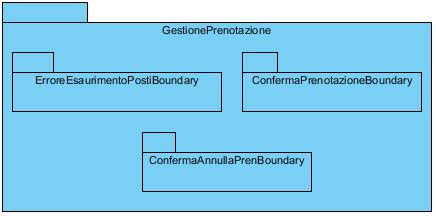
**2.2.3 View**

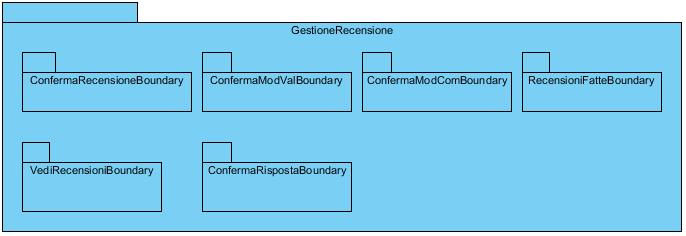
Il sottopackage “View” è presentato nel seguente schema e contiene le classi Java che si occupano della logica di presentazione del sistema.

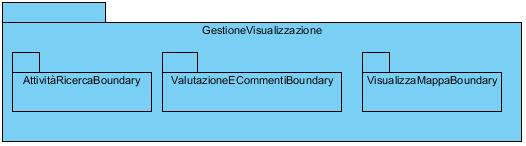


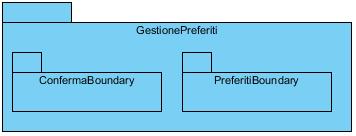


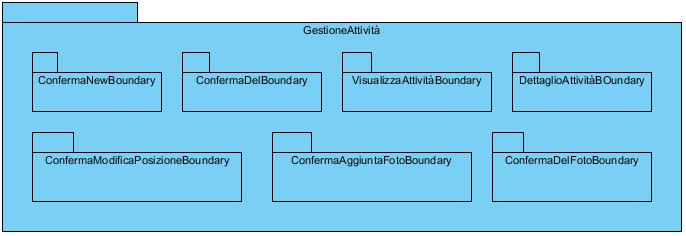


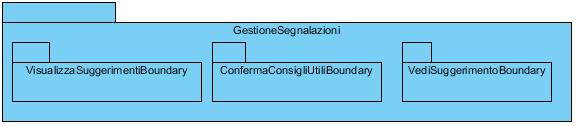


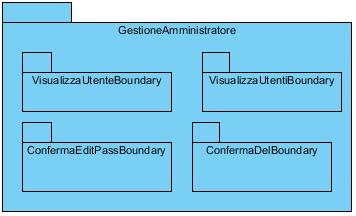












# **3. Interfacce delle Classi**

|  |  |
| --- | --- |
| **RecensioneDAO** | |
| **Servizio** | **Descrizione** |
| public void doSave(RecensioneBean recensione) | Il sottosistema permette di effettuare una recensione ad una attività ad un utente, compilando i campi della valutazione e del commento.  L’oggetto recensione passato come parametro verrà salvato nel database |
| public void doDelete(RecensioneBean recensione) | Il sottosistema permette di cancellare una recensione effettuata da un utente ad una attività.  L’ oggetto recensione passato come parametro verrà eliminato dal database. |
| public List<RecensioneBean> doRetrieveAll() | Il sottosistema permette di visualizzare tutte le recensioni effettuate.  La lista di recensioni passati come parametro verrà prelevato dal database |
| public Utente doRetrieveByKey(String username, int IDAttivita) | Il sottosistema permette di recuperare i dati relativi a una specifica recensione nel sito.  Username e IDAttività sono le chiavi utilizzate per prelevare una determinata recensione dal database. |
| Public void doUpdate (RecensioneBean recensione) | Il sottosistema permette di modificare valutazione o commento di una recensione già presente nel sistema. |

|  |  |
| --- | --- |
| **PrenotazioneDAO** | |
| **Servizio** | **Descrizione** |
| public void doSave(PrenotazioneBean prenotazione) | Il sottosistema permette di effettuare una prenotazione ad una attività ad un utente, compilando i campi del giorno, dell’ora e dei posti.  L’oggetto prenotazione passato come parametro verrà salvato nel database |
| public void doDelete(PrenotazioneBean prenotazione) | Il sottosistema permette di annullare una prenotazione effettuata da un utente ad una attività.  L’ oggetto prenotazione passato come parametro verrà eliminato dal database. |
| public List<PrenotazioneBean> doRetrieveAll() | Il sottosistema permette di visualizzare tutte le prenotazioni effettuate.  La lista di prenotazioni verrà prelevato dal database. |
| public PrenotazioneBean doRetrieveByKey(String username, int IDAttività) | Il sottosistema permette di recuperare i dati relativi a una specifica prenotazione nel sito.  Username e IDAttività sono le chiavi utilizzate per prelevare una determinata prenotazione dal database. |
| Public void doUpdate (PrenotazioneBean prenotazione) | Il sottosistema permette di modificare data, ora o posti di una prenotazione già presente nel sistema. |

|  |  |
| --- | --- |
| **PreferitiDAO** | |
| **Servizio** | **Descrizione** |
| public void doSave(PreferitiBean preferiti) | Il sottosistema permette di aggiungere alla lista preferiti una attività ad un utente.  L’oggetto preferiti passato come parametro verrà salvato nel database |
| public void doDelete(PreferitiBean preferiti) | Il sottosistema permette di rimuovere una attività dalla lista preferiti dell’utente.  L’ oggetto preferiti passato come parametro verrà eliminato dal database. |
| public List<PreferitiBean> doRetrieveAll() | Il sottosistema permette di visualizzare tutte le attività aggiunte alla lista preferiti.  La lista dei preferiti verrà prelevata dal database. |
| public PreferitiBean doRetrieveByKey(String username, int IDAttività) | Il sottosistema permette di recuperare i dati relativi a una specifica attività nella lista preferiti.  Username e IDAttività sono le chiavi utilizzate per prelevare una determinata attività dal database. |
| Public void doUpdate (PreferitiBean preferiti) | Il sottosistema permette di modificare una attività presente nella lista preferiti già presente nel sistema. |

|  |  |
| --- | --- |
| **PersonaDAO** | |
| **Servizio** | **Descrizione** |
| public void doSave(PersonaBean persona) | Il sottosistema permette di registrare uno utente nel sistema attraverso la compilazione di un apposito form.  L’oggetto utente passato come parametro verrà salvato nel database |
| public void doDelete(PersonaBean persona) | Il sottosistema permette di cancellare un account utente. L’amministratore può eliminare un utente specifico dal sito.  L’ oggetto utente passato come parametro verrà eliminato dal database. |
| public List< PersonaBean persona > doRetrieveAll() | Il sottosistema permette di visualizzare i dati degli utenti del sito. L’amministratore può visualizzare tutti gli utenti registrati al sito.  La lista di tutti gli utenti verrà prelevata dal database e visualizzata. |
| public PersonaBean doRetrieveByKey(String idPersona) | Il sottosistema permette di recuperare i dati relativi a un utente registrato nel sito cliccando su uno specifico utente nella lista degli utenti registrati nel sito, e li visualizza.  L’idPersona è il codice univoco passato come parametro per prelevare i dati dell’ utente dal database. |
| public void doUpdate (PersonaBean persona) | Il sottosistema permette di modificare i dati di un utente registrato nel sistema. L’utente nella gestione dei dati personali può modificare i suoi dati personali. |

|  |  |
| --- | --- |
| **AttivitàDAO** | |
| **Servizio** | **Descrizione** |
| public void doSave(AttivitàBean attività) | Il sottosistema permette di registrare un’attività nel sistema dal ristoratore attraverso la compilazione di un apposito form.  L’oggetto utente passato come parametro verrà salvato nel database |
| public void doDelete(AttivitàBean attività) | Il sottosistema permette di cancellare un’attività dal sistema dal ristoratore o dall’amministratore .  L’ oggetto attività passato come parametro verrà eliminato dal database. |
| public List<AttivitàBean> doRetrieveAll() | Il sottosistema permette di visualizzare i dati delle attività del sito. L’amministratore può visualizzare tutte le attività registrate al sito.  La lista di tutte le attività verrà prelevata dal database e visualizzata. |
| public List<AttivitàBean> doRetrievebyRistoratore(RistoratoreBean ristoratore) | Il sottosistema permette di visualizzare i dati delle attività del sito di uno specifico ristoratore. Il ristoratore può visualizzare tutte le sue attività registrate al sito.  La lista di tutte le attività utenti verrà prelevata dal database e visualizzata. |
| public AttivitàBean doRetrieveByKey(String idAttività) | Il sottosistema permette di recuperare i dati relativi a un‘attività.  L’idAttività è il codice univoco passato come parametro per prelevare i dati di un’attività dal database. |
| public void doUpdate (AttivitàBean attività) | Il sottosistema permette di modificare i dati di un’attività registrata nel sistema. Il ristoratore nella gestione dei dati delle sue attività può modificare i dati delle sue attività. |

|  |  |
| --- | --- |
| **PiattiDAO** | |
| **Servizio** | **Descrizione** |
| public void doSave(PiattiBean piatto) | Il sottosistema permette di registrare un piatto nel sistema. L’oggetto piatto passato come parametro verrà salvato nel database. |
| public void doDelete(PiattiBean piatto) | Il sottosistema permette di cancellare un piatto.  L’ oggetto piatto passato come parametro verrà eliminato dal database. |
| public List<PiattiBean> doRetrieveAll() | Il sottosistema permette di visualizzare i piatti del sito.  La lista dei piatti verrà prelevata dal database. |
| public PiattiBean doRetrieveByKey(String idPiatti) | Il sottosistema permette di recuperare i dati relativi a un piatto registrato nel sito.  L’attributo idPiatti, è il codice univoco passato come parametro per prelevare i dati del piatto dal database |
| public PiattiBean doUpdate (PiattiBean foto) | Il sottosistema permette di aggiornare un piatto registrata nel sito.  L’attributo piatto passato come parametro è il piatto che vuoi modificare. |

|  |  |
| --- | --- |
| **FotoDAO** | |
| **Servizio** | **Descrizione** |
| public void doSave (FotoBean foto) | Il sottosistema permette di registrare una foto nel sistema. L’oggetto foto passato come parametro verrà salvato nel database |
| public void doDelete(FotoBean foto) | Il sottosistema permette di cancellare una foto.  L’ oggetto foto passato come parametro verrà eliminato dal database. |
| public List<FotoBean> doRetrieveAll() | Il sottosistema permette di visualizzare i dati delle foto del sito.  La lista degli operatori verrà prelevata dal database. |
| public FotoBean doRetrieveByKey(String idFoto) | Il sottosistema permette di recuperare i dati relativi a una foto registrata nel sito.  L’attributo idfoto è il codice univoco passato come parametro per prelevare i dati della foto dal database. |
| public FotoBean doUpdate (FotoBean foto) | Il sottosistema permette di aggiornare una foto registrata nel sito.  L’attributo foto passato come parametro è la foto che vuoi modificare. |

|  |  |
| --- | --- |
| **SuggerimentoDAO** | |
| **Servizio** | **Descrizione** |
| public void doSave(SuggerimentoBean segnalazione) | Il sottosistema permette di registrare una suggerimento nel sistema. L’oggetto suggerimento passato come parametro verrà salvato nel database |
| public void doDelete (SuggerimentoBean segnalazione) | Il sottosistema permette di cancellare un suggerimento.  L’ oggetto Suggerimento passato come parametro verrà eliminato dal database. |
| public List<SuggerimentoBean> doRetrieveAll() | Il sottosistema permette di visualizzare i dati dei suggerimenti del sito.  La lista dei suggerimenti verrà prelevato dal database. |
| public SuggerimentoBean doRetrieveByKey(String idSegnalazione) | Il sottosistema permette di recuperare i dati relativi a un suggerimento registrato nel sito.  L’ attributo id Segnalazioni è il codice univoco passato come parametro per prelevare i dati della Segnalazione dal database. |
| public SuggerimentoBean doUpdate(SuggerimentoBean) | Il sottosistema permette di aggiornare un suggerimento registrato nel sito.  L’attributo Suggerimento passato come parametro è il suggerimento che vuoi modificare. |