Object Design Document (ODD)

Corso di Ingegneria del software a.a. 2016/17
Università degli Studi di Salerno

NOME: Altieri Carmine
MATRICOLA: 0512101882

Indice

INTRODUZIONE

TRADE OFF

Comprensibilità VS Costi

Si preferisce aggiungere costi per la documentazione al fine di rendere il codice comprensibile anche alle persone non coinvolte nel progetto o le persone coinvolte che non hanno lavorato a quella parte in particolare. Commenti diffusi nel codice facilitano la comprensione, di conseguenza migliorare la comprensibilità agevola il mantenimento e anche il processo di modifica.

Interfaccia VS Easy-use

Il sistema VLT è molto semplice e di facile utilizzo poiché ha un'interfaccia chiara e intuitiva.

LINEE GUIDA PER LA DOCUMENTAZIONE DELL'INTERFACCIA

La convenzione per quanto riguarda i nomi dei file, delle operazioni e delle variabili, è la nota Camel Notation.

Organizzazione dei file

Ogni file deve essere:

- Sviluppato e diviso in base alla categoria di appartenenza, ovvero deve essere correlato ad un'unica funzionalità che persegue. Ogni pagina di VLT (login, registrazione, modifica profilo, visualizza evento etc.) deve essere implementata in file separati;
- Diviso in più file, se raggiunge una lunghezza tale da divenire difficile da leggere e comprendere;

La convenzione per quanto riguarda i nomi dei file, delle operazioni e delle variabili è quella di avere nomi evocativi, ma soprattutto in lingua italiana.

Vengono utilizzate una serie di script/pagine web i cui nomi hanno come prefisso "recupera" o "controlla". Tali pagine web prelevano i dati dal client e invocano le funzionalità dei gestori creando opportuni oggetti o passando in input i dati grezzi.

Spostamento di linee

Quando un'espressione supera la lunghezza della linea, occorre spezzarla secondo i seguenti principi generali:

- Interrompere la linea dopo una virgola;
- Interrompere la linea prima di un operatore;
- Preferire interruzioni di alto livello rispetto ad interruzioni di basso livello (interrompere laddove non si interrompe un discorso logico, discorso valido soprattutto per le formule es. (3+4) * 2 interrompere prima della moltiplicazione senza spezzare gli operandi in parentesi);
- Allineare la nuova linea con l'inizio dell'espressione nella linea precedente;
- Se le regole precedenti rendono il codice più confuso o il codice è troppo spostato verso il margine destro, utilizzare solo otto spazi di indentazione.

Indentazione

L'indentazione deve essere effettuata con un TAB e qualunque sia il linguaggio usato per la produzione di codice, ogni istruzione deve essere opportunamente indentata.

Es.

<html>

<head>

</head>

<body>

</body>

</html>

Deve essere sostituita da:

<html>

    <head>

    </head>

    <body>

    </body>

</html>

Questa pratica deve essere usata soprattutto per le istruzioni FOR, IF.

È buona pratica scendere di livello.

Inizializzazione

Inizializzare le variabili locali nel punto in cui sono state dichiarate a meno che il suo valore iniziale non dipenda da un calcolo che occorre eseguire prima.

Posizione

Mettere le dichiarazioni all'inizio dei blocchi. Evitare per quanto possibile di dichiarare le variabili al loro primo uso: può confondere il programmatore inesperto e impedire la portabilità del codice dentro lo scope. L'unica eccezione a questa regola sono gli indici dei cicli for che in Java possono essere dichiarati nell' istruzione stessa. Evitare dichiarazioni locali che nascondono dichiarazioni a più alto livello. Ad esempio, evitare di dichiarare una variabile con lo stesso nome in un blocco interno.

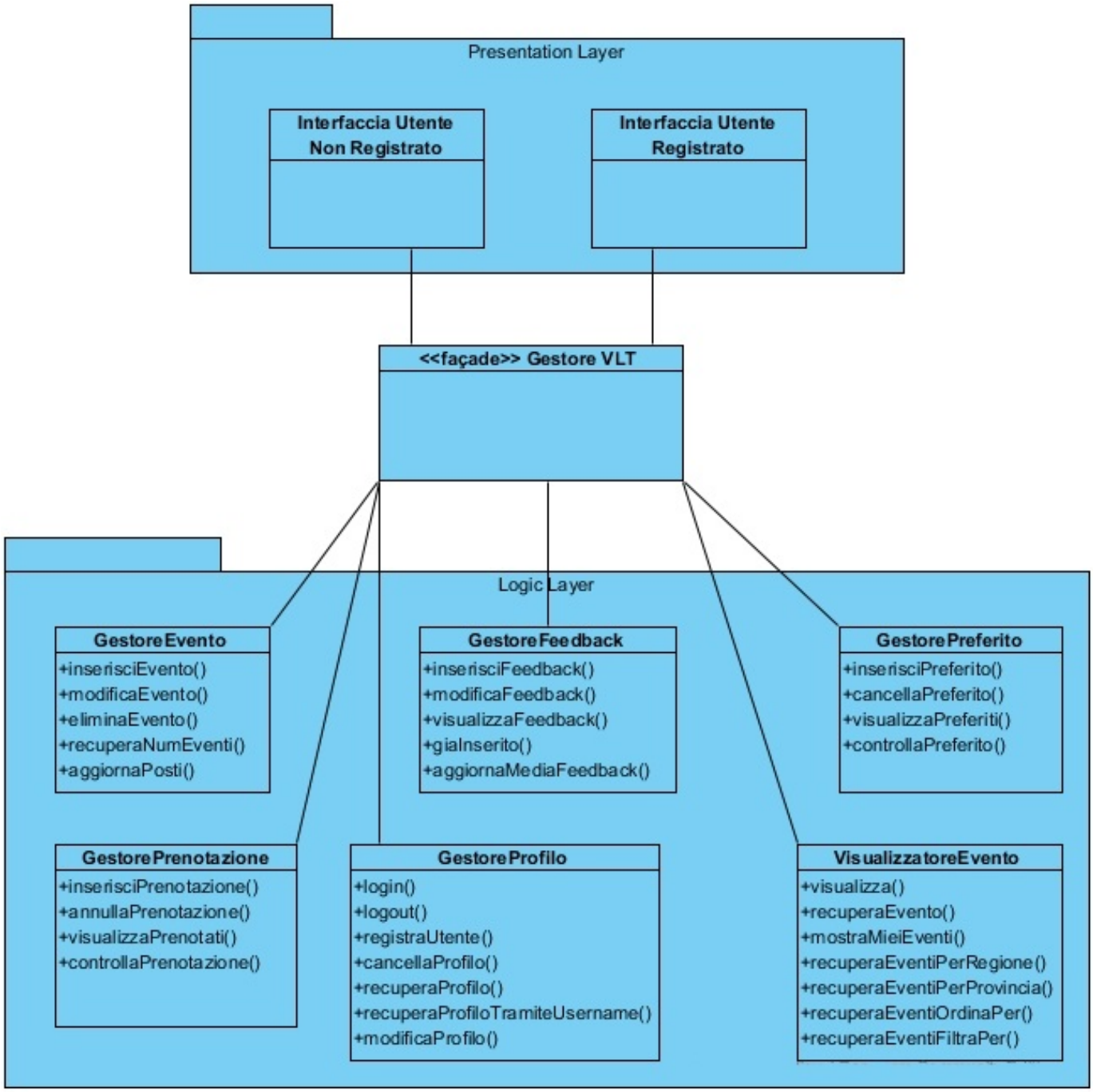
Parentesi

A prescindere dalle istruzioni che seguono un IF, è necessario, laddove ci fosse anche una sola istruzione, riportare il blocco di istruzioni tra parentesi graffe. Ogni tag di apertura deve essere necessariamente seguito dall'apposito tag di chiusura (eccetto i tag self-closing).

Una convenzione importante, per quanto riguarda l'inserimento di numeri o di valori costanti, è quella di non usare una codifica fissa (hard coding) che è fortemente sconsigliata ma di associare sempre il valore ad una variabile o semplicemente definire una macro che può essere richiamata da eventi ed essere parametrizzata. In questo modo si facilita la modifica, sostituendo solo il valore della variabile o macro, in un unico posto.

DESIGN PATTERN

Façade Pattern



VLT fa uso del Façade Pattern per definire un’unica interfaccia a livello di logica che permette all’utente di interagire, attraverso l’interfaccia, con le funzionalità del sistema vedendole come un unico sistema.

DEFINIZIONI, ACRONIMI E ABBREVIAZIONI

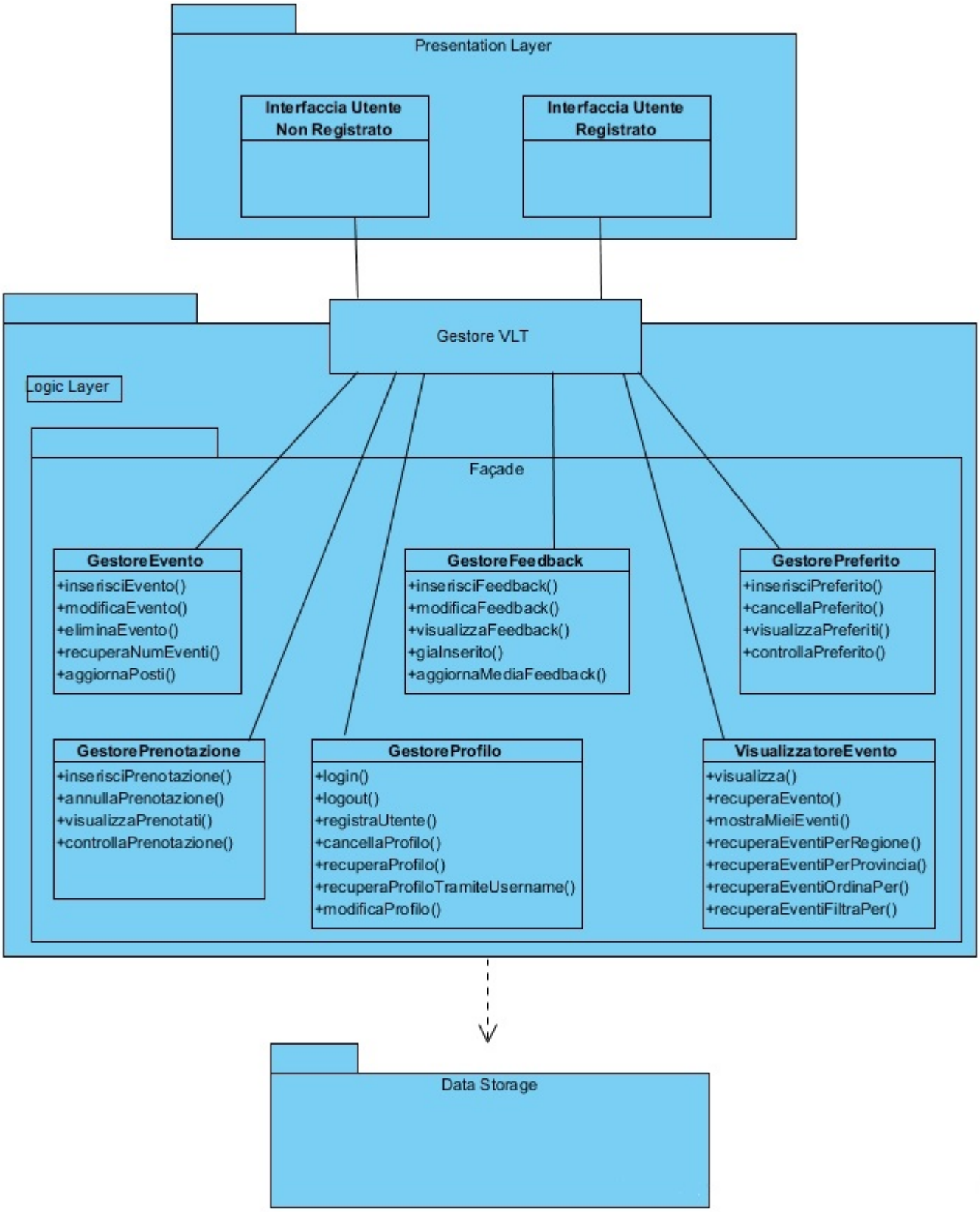
Acronimo	Definizione
DBMS	Database Management System
HTML	Linguaggio di mark-up per pagine web
CSS	Linguaggio usato per definire la formattazione di pagine web
JAVASCRIPT	Linguaggio di scripting utilizzato lato client per la dinamicità del portale
Camel Notation	Consiste nello scrivere più parole insieme delimitando la fine e l'inizio di una nuova parola con una lettera maiuscola
PHP	PHP HypertextPreProcessor (linguaggio di scripting utilizzato lato server)
DB	Database
QUERY	Interrogazione al database
VLT	Vinyl Listening Together (nome del portale web)
Altervista	Piattaforma web gratuita utilizzata per il servizio di hosting di VLT

RIFERIMENTI

L'insieme del materiale di riferimento utilizzato per la realizzazione del progetto e per la stesura di questo stesso documento comprende:

- Slide del docente, reperibili sulla piattaforma "e-learning" (http://elearning.informatica.unisa.it/el-platform/).
- Libro di testo: Bruegge, A.H. Dutoit, Object Oriented Software Engineering.
- Documento SDD di VLT.

PACKAGES


Presentation Layer
Interfaccia Utente Non Registrato
Interfaccia Utente Registrato
Gestore VLT
Logic Layer
Façade
GestoreEvento
+inserisciEvento()
+modificaEvento()
+eliminaEvento()
+recuperaNumEventi()
+aggiornaPosti()
GestoreFeedback
+inserisciFeedback()
+modificaFeedback()
+visualizzaFeedback()
+giaInserito()
+aggiornaMediaFeedback()
GestorePreferito
+inserisciPreferito()
+cancellaPreferito()
+visualizzaPreferiti()
+controllaPreferito()
GestorePrenotazione
+inserisciPrenotazione()
+annullaPrenotazione()
+visualizzaPrenotati()
+controllaPrenotazione()
GestoreProfilo
+login()
+logout()
+registraUtente()
+cancellaProfilo()
+recuperaProfilo()
+recuperaProfiloTramiteUsername()
+modificaProfilo()
VisualizzatoreEvento
+visualizza()
+recuperaEvento()
+mostraMieiEventi()
+recuperaEventiPerRegione()
+recuperaEventiPerProvincia()
+recuperaEventiOrdinaPer()
+recuperaEventiFiltraPer()
Data Storage

INTERFACCE DELLE CLASSI

Nome classe	GestoreEvento
Descrizione	Rappresenta il gestore delle funzionalità relate agli eventi quale inserimento, modifica, cancellazione e operazioni ausiliarie come il recupero del numero degli eventi di un utente e aggiornamento dei posti disponibili di un evento.
Pre-condizione	**context** GestoreEvento:: inserisciEvento(Evento); **pre:** Evento!=null **context** GestoreEvento:: modificaEvento(Evento, Id); **pre:** Evento!=null && Id!=null **context** GestoreEvento:: eliminaEvento(Id); **pre:** Id!=null **context** GestoreEvento:: recuperaNumEventi(Email); **pre:** Email!=null **context** GestoreEvento:: aggiornaPosti(NewPosti, Id); **pre:** NewPosti!=null && Id!=null
Post-condizione	**context** GestoreEvento:: inserisciEvento(Evento); **post:** evento inserito **context** GestoreEvento:: modificaEvento(Evento, Id); **post:** Evento=newEvento **context** GestoreEvento:: eliminaEvento(Id); **post:** evento eliminato **context** GestoreEvento:: recuperaNumEventi(Email); **post:** numero di eventi inseriti dall'utente **context** GestoreEvento:: aggiornaPosti(NewPosti, Id); **post:** Posti=NewPosti (relativo all'evento indicato in Id)

Nome classe	GestoreFeedback
Descrizione	Gestisce le funzionalità relative al sistema feedback quali inserimento, modifica, visualizzazione o operazioni ausiliarie come il controllo di un feedback già inserito e l'aggiornamento della media feedback degli utenti
Pre-condizione	**context** GestoreFeedback:: inserisciFeedback(Feedback); **pre:** Feedback.votante!=null && Feedback.votato!=null && Feedback.votazione!=null && Feedback.votazione>=1 && Feedback.votazione<=5 **context** GestoreFeedback:: modificaFeedback(Feedback); **pre:** Feedback.votante!=null && Feedback.votato!=null && Feedback.votazione!=null && Feedback.votazione>=1 && Feedback.votazione<=5 **context** GestoreFeedback:: visualizzaFeedback(Email); **pre:** Email!=null **context** GestoreFeedback:: giaInserito(Votante, Votato); **pre:** Votante!=null && Votato!=null **context** GestoreFeedback:: aggiornaMediaFeedbackUtente(Votato); **pre:** Votato!=null
Post-condizione	**context** GestoreFeedback:: inserisciFeedback(Feedback); **post:** feedback inserito || feedback modificato **context** GestoreFeedback:: modificaFeedback(Feedback); **post:** feedback feedback modificato **context** GestoreFeedback:: visualizzaFeedback(Email); **post:** elenco feedback (relativo all'utente indicato in Email) **context** GestoreFeedback:: giaInserito(Votante, Votato); **post:** dà come risultato il valore booleano 'true' se il feedback esiste, 'false' altrimenti **context** GestoreFeedback:: aggiornaMediaFeedbackUtente(Votato); **post:** media feedback aggiornata (relativa all'utente indicato in Votato)

Nome classe	GestorePreferiti
Descrizione	Gestisce le funzionalità relative al sistema di gestione degli utenti preferiti quali inserimento di un utente ai preferiti, cancellazione dello stesso, visualizzazione dei preferiti e operazione di controllo di utente preferito già esistente
Pre-condizione	**context** GestorePreferiti:: inserisciPreferito(EmailUtente, EmailPreferito); **pre:** EmailUtente!=null && EmailPreferito!=null **context** GestorePreferiti:: cancellaPreferito(EmailUtente, EmailPreferito); **pre:** EmailUtente!=null && EmailPreferito!=null **context** GestorePreferiti:: visualizzaPreferiti(EmailUtente); **pre:** EmailUtente!=null **context** GestorePreferiti:: controllaPreferito(EmailUtente, EmailPreferito); **pre:** EmailUtente!=null && EmailPreferito!=null
Post-condizione	**context** GestorePreferiti:: inserisciPreferito(EmailUtente, EmailPreferito); **post:** utente inserito tra i preferiti (indicato in EmailPreferito) **context** GestorePreferiti:: cancellaPreferito(EmailUtente, EmailPreferito); **post:** utente cancellato tra i preferiti (indicato in EmailPreferito) **context** GestorePreferiti:: visualizzaPreferiti(EmailUtente); **post:** elenco preferiti (relativi a utente indicato in EmailUtente) **context** GestorePreferiti:: controllaPreferito(EmailUtente, EmailPreferito); **post:** dà come risultato il valore booleano ‘true’ se l’utente preferito già è presente, ‘false’ altrimenti

Nome classe	GestorePrenotazione
Descrizione	Gestisce le funzionalità relative al sistema di prenotazione degli eventi da parte degli utenti quali l’inserimento, l’annullamento, la visualizzazione delle prenotazioni e operazione di controllo di una prenotazione già esistente
Pre-condizione	**context** GestorePrenotazione:: inserisciPrenotazione(Id, EmailUtente); **pre:** Id!=null && EmailUtente!=null **context** GestorePrenotazione:: annullaPrenotazione(Id, EmailUtente); **pre:** Id!=null && EmailUtente!=null **context** GestorePrenotazione:: visualizzaPrenotati(Id); **pre:** Id!=null **context** GestorePrenotazione:: controllaPrenotazione(Id, EmailUtente); **pre:** Id!=null && EmailUtente!=null
Post-condizione	**context** GestorePrenotazione:: inserisciPrenotazione(Id, EmailUtente); **post:** prenotazione all’evento (indicato in Id) fatta da utente (indicato in EmailUtente) **context** GestorePrenotazione:: annullaPrenotazione(Id, EmailUtente); **post:** prenotazione all’evento (indicato in Id) annullata da utente (indicato in EmailUtente) **context** GestorePrenotazione:: visualizzaPrenotati(Id); **post:** elenco utenti prenotati all’evento (indicato in Id) **context** GestorePrenotazione:: controllaPrenotazione(Id, EmailUtente); **post:** dà come risultato il valore booleano ‘true’ se l’utente si è già prenotato all’evento (indicato in Id), ‘false’ altrimenti

Nome classe	GestoreProfilo
Descrizione	Gestisce le funzionalità relative agli utenti quali login, logout, registrazione, cancellazione di un profilo, modifica profilo, recupero di un profilo (tramite username o indirizzo email)
Pre-condizione	**context** GestoreProfilo:: login(Email, Pass); **pre:** Email!=null && Pass!=null **context** GestoreProfilo:: logout(); **pre:** utente deve essere loggato **context** GestoreProfilo:: registraUtente(Utente); **pre:** Utente.email!=null && Utente.username!=null && Utente.password!=null **context** GestoreProfilo:: cancellaProfilo(Email); **pre:** Email!=null && utente deve essere loggato **context** GestoreProfilo:: modificaProfilo(Email, NewEmail, NewUsername, NewPassword); **pre:** Email!=null && NewEmail!=null && NewUsername!=null && NewPassword!=null **context** GestoreProfilo:: recuperaProfilo(Email); **pre:** Email!=null **context** GestoreProfilo:: recuperaProfiloTramiteUsername(Username); **pre:** Username!=null
Post-condizione	**context** GestoreProfilo:: login(Email, Pass); **post:** dà il valore booleano 'false' (mancata corrispondenza utente), utente viene altrimenti loggato **context** GestoreProfilo:: logout(); **post:** utente disconnesso (e rediretto alla home) **context** GestoreProfilo:: registraUtente(Utente); **post:** utente registrato **context** GestoreProfilo:: cancellaProfilo(Email); **post:** utente cancellato **context** GestoreProfilo:: modificaProfilo(Email, NewEmail, NewUsername, NewPassword); **post:** Email=NewEmail && Username=NewUsername && Password=NewPassword (dati relativi all'utente indicato in Email) **context** GestoreProfilo:: recuperaProfilo(Email); **post:** restituzione dell'utente (con relative info profilo) indicato in Email **context** GestoreProfilo:: recuperaProfiloTramiteUsername(Username); **post:** restituzione dell'utente (con relative info profilo) indicato in Username

Nome classe	VisualizzatoreEvento
Descrizione	Gestisce le funzionalità relative al sistema di visualizzazione e recupero degli eventi quali la visualizzazione, il recupero degli eventi (per regione o per provincia) e ordinamento e filtraggio degli stessi.
Pre-condizione	**context** VisualizzatoreEvento:: visualizza(Id); **pre:** Id!=null **context** VisualizzatoreEvento:: recuperaEvento(Ris); **pre:** Ris!=null **context** VisualizzatoreEvento:: mostraMieiEventi(Email); **pre:** Email!=null **context** VisualizzatoreEvento:: recuperaEventiPerRegione(Regione); **pre:** Regione!=null **context** VisualizzatoreEvento:: recuperaEventiPerProvincia(Provincia); **pre:** Provincia!=null **context** VisualizzatoreEvento:: recuperaEventiOrdinaPer(Criterio, Cerca); **pre:** Criterio="Data" || Criterio="Ingresso" && Cerca=nomeregione || Cerca=nomeprovincia **context** VisualizzatoreEvento:: recuperaEventiFiltraPer(Criterio, Cerca, Email); **pre:** Criterio="Gratuito" || Criterio="Preferiti" && Cerca=nomeregione || Cerca=nomeprovincia && Email!=null
Post-condizione	**context** VisualizzatoreEvento:: visualizza(Id); **post:** restituzione dell'evento (con relative info) indicato in Id **context** VisualizzatoreEvento:: mostraMieiEventi(Email); **post:** elenco eventi creati relativi all'utente indicato in Email **context** VisualizzatoreEvento:: recuperaEventiPerRegione(Regione); **post:** elenco eventi in cui Regione='nomeregione' **context** VisualizzatoreEvento:: recuperaEventiPerProvincia(Provincia); **post:** elenco eventi in cui Provincia='nomeprovincia'

	context VisualizzatoreEvento:: recuperaEventiOrdinaPer(Criterio, Cerca); **post:** elenco eventi in cui Cerca=’nomeprovincia’ || Cerca=’nomeregione’ && ordinati in base al criterio indicato in Criterio **context** VisualizzatoreEvento:: recuperaEventiFiltraPer(Criterio, Cerca, Email); **post:** elenco eventi in cui Cerca=’nomeprovincia’ || Cerca=’nomeregione’ && fitrati in base al criterio indicato in Criterio && relativi a utente indicato in Email

GLOSSARIO

Feedback: votazione espressa in scala da 1 a 5 visualizzato graficamente tramite stelle.
Form: scheda di compilazione per l’inserimento di dati, dove l’utente inserisce e invia i dati al server.
Organizzatore: l’utente registrato che ha creato l’evento.
Login: procedura di accesso (autenticazione) ad un sistema/applicazione informatica con le credenziali registrate al sito web.
Logout: procedura di uscita da un sistema/applicazione informatica.
Password: parola chiave con cui l’utente accede al sito web.
Registrazione: procedura di salvataggio dei dati dell’utente nel database con i quali accede alle funzionalità che offre il sistema.
Sito web: insieme di pagine correlate tra loro.
Utente: utente generico, che si divide in utente non registrato e utente registrato.
Utente non registrato: utente che usufruisce di limitate funzionalità del sito in quanto non registrato.
Utente registrato: utente che usufruisce di tutte le funzionalità del sito in quanto registrato (dopo aver effettuato il login).
Partecipante: utente registrato che si è prenotato ad un evento.
DB: sistema di memorizzazione di dati permanenti.