Object Design Document

FormulaOnline

Versione	1.0
Presentato da	ALFONSO ANZELMO, PAOLO D'ANTUONO

Storico versioni

Data	Versione	Descrizione	Autori
22/02/2023	0.1	Prima stesura	A. Anzelmo, P. D'Antuono
20/06/2024	0.2	Modifica Sezione 1, aggiunta dei package	A. Anzelmo, P. D'Antuono
25/06/2024	0.3	Aggiunta class interfaces	A. Anzelmo, P. D'Antuono
27/06/2024	0.4	Aggiunta design pattern, aggiunta glossario	A. Anzelmo, P. D'Antuono
05/07/2024	0.5	Modifiche sui package Modifica campo Segnalazione	A. Anzelmo, P. D'Antuono
17/07/2024	0.6	Correzione package e modello e/r ristrutturato	A. Anzelmo, P. D'Antuono
18/07/2024	1.0	Revisione finale e aggiunta links	A. Anzelmo, P. D'Antuono

Sommario

1. Introduzione

FormulaOnline vuole creare una comunità di appassionati di Formula 1 in Italia, permettendo loro di discutere di questo sport, esprimendo opinioni su questa passione.

Questo documento ha l'obiettivo di produrre un modello che integra tutte le informazioni raccolte nella fase di analisi e di system design così da passare alla fase di implementazione.
In questa prima sezione vi sono le definizioni, acronimi utilizzati, riferimenti ad altri documenti.

1.1. Object Design trade off

Tempo di rilascio vs. funzionalità
Al fine di rispettare i tempi di rilascio si è deciso di non sviluppare tutte le funzionalità.

Sicurezza vs. Tempo di risposta
Al fine di avere una maggiore sicurezza poiché la comunicazione tra utenti avviene in maniera asincrona.

1.2. Linee guida per la documentazione dell'interfaccia

Gli sviluppatori dovrebbero seguire una serie di regole durante la progettazione delle interfacce.
Si fa riferimento a:

- Java Sun: https://checkstyle.sourceforge.io/sun_style.html
- HTML: https://www.w3schools.com/html/html5_syntax.asp
- JSP : Code Conventions for the JavaServer Pages Technology Version 1.x Language (oracle.com)

1.3. Definizioni, acronimi e abbreviazioni

JSP: Java Server Pages una la tecnologia per integrare HTML con codice java.

Package: è un insieme di classi raccolte in un pacchetto.

Design pattern: è un template di soluzioni a problemi ricorrenti, sperimentati e approvati per avere riuso e flessibilità.

Javadoc: è un tool che permette di produrre una pagina html in automatico, commentando la classe con apposite keyword.

interfaccia: è la lista delle operazioni che una classe offre.

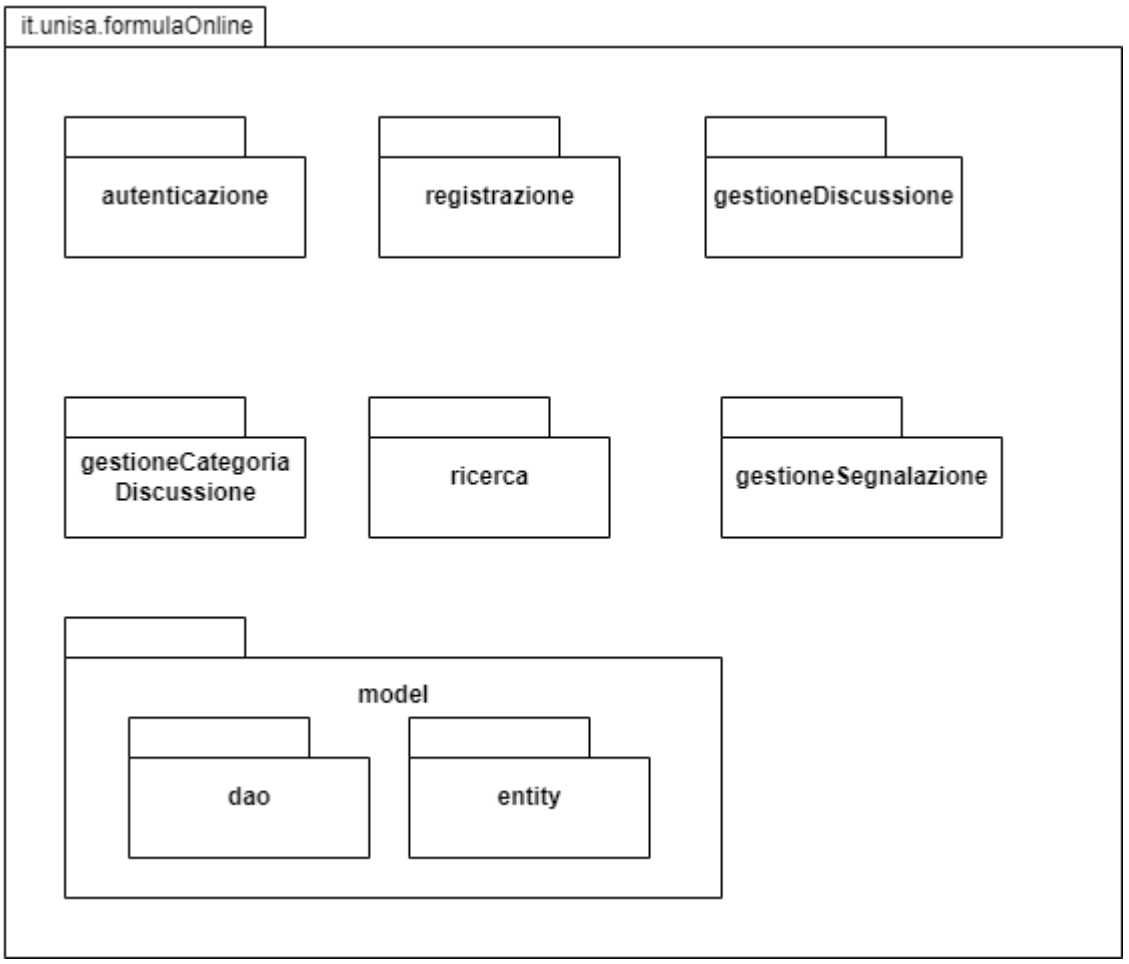
1.4. Riferimenti

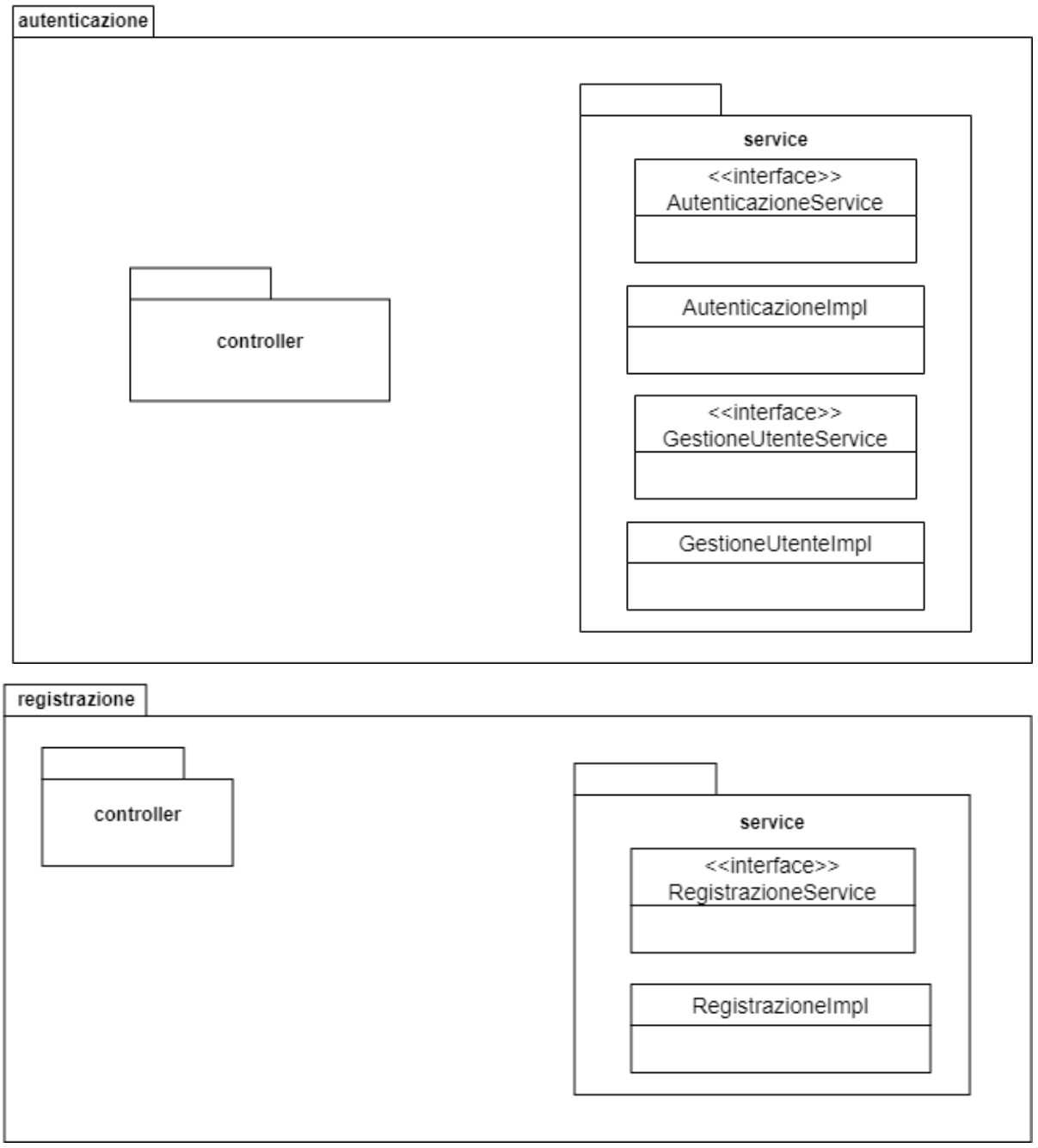
- Statement Of Work
- Requirements Analysis Document
- System Design Document
- Test Plan
- Test Case Specification
- Test Incident Report
- Test Summary Report
- Matrice di tracciabilità
- Manuale di installazione
- Manuale utente

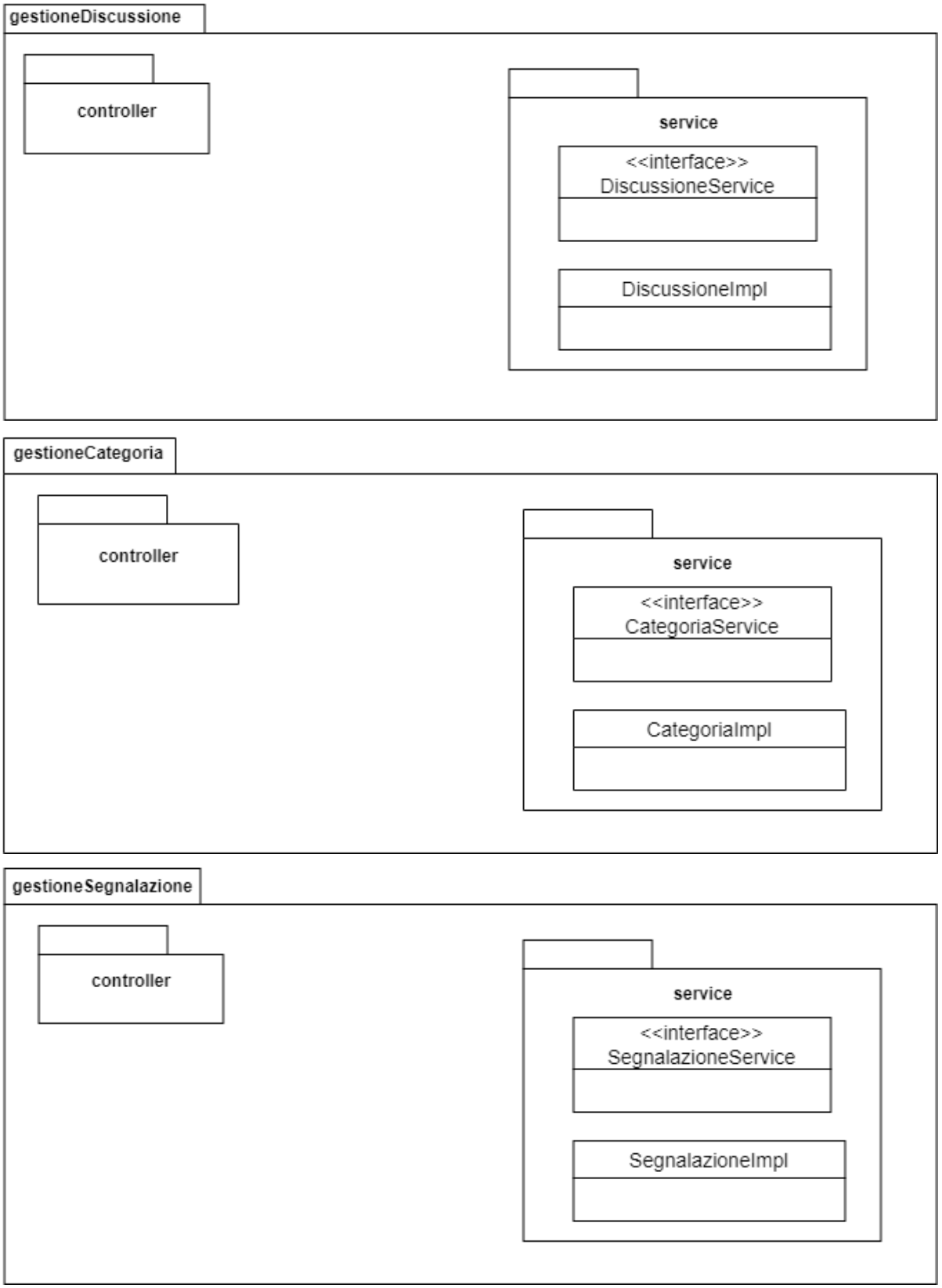
2. Packages

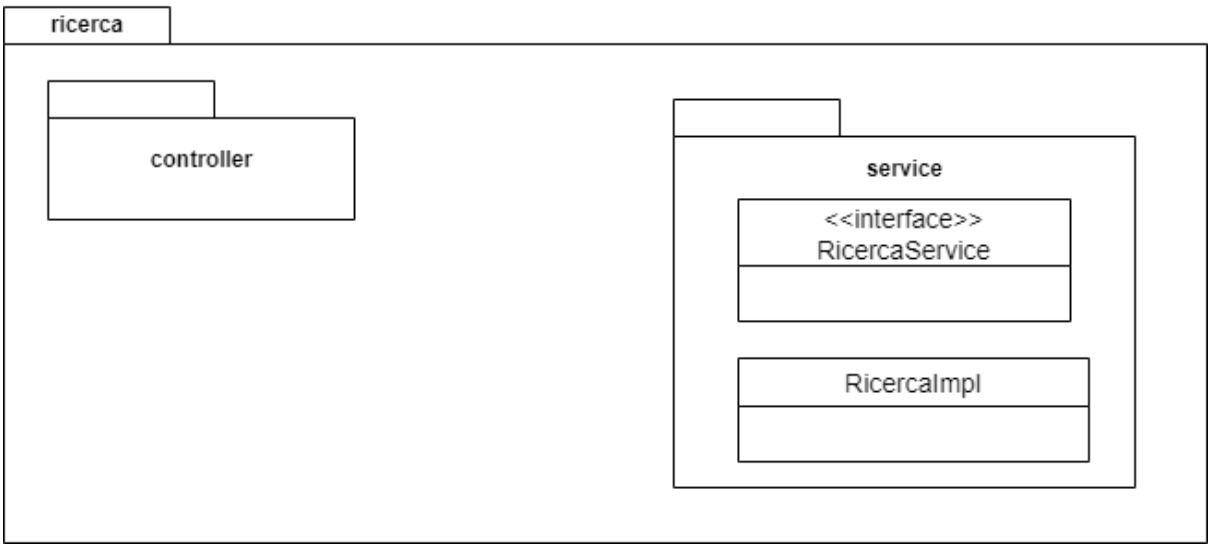
Questa sezione mostra i package di FormulaOnline che utilizza i sistemi definiti nella parte di system design.

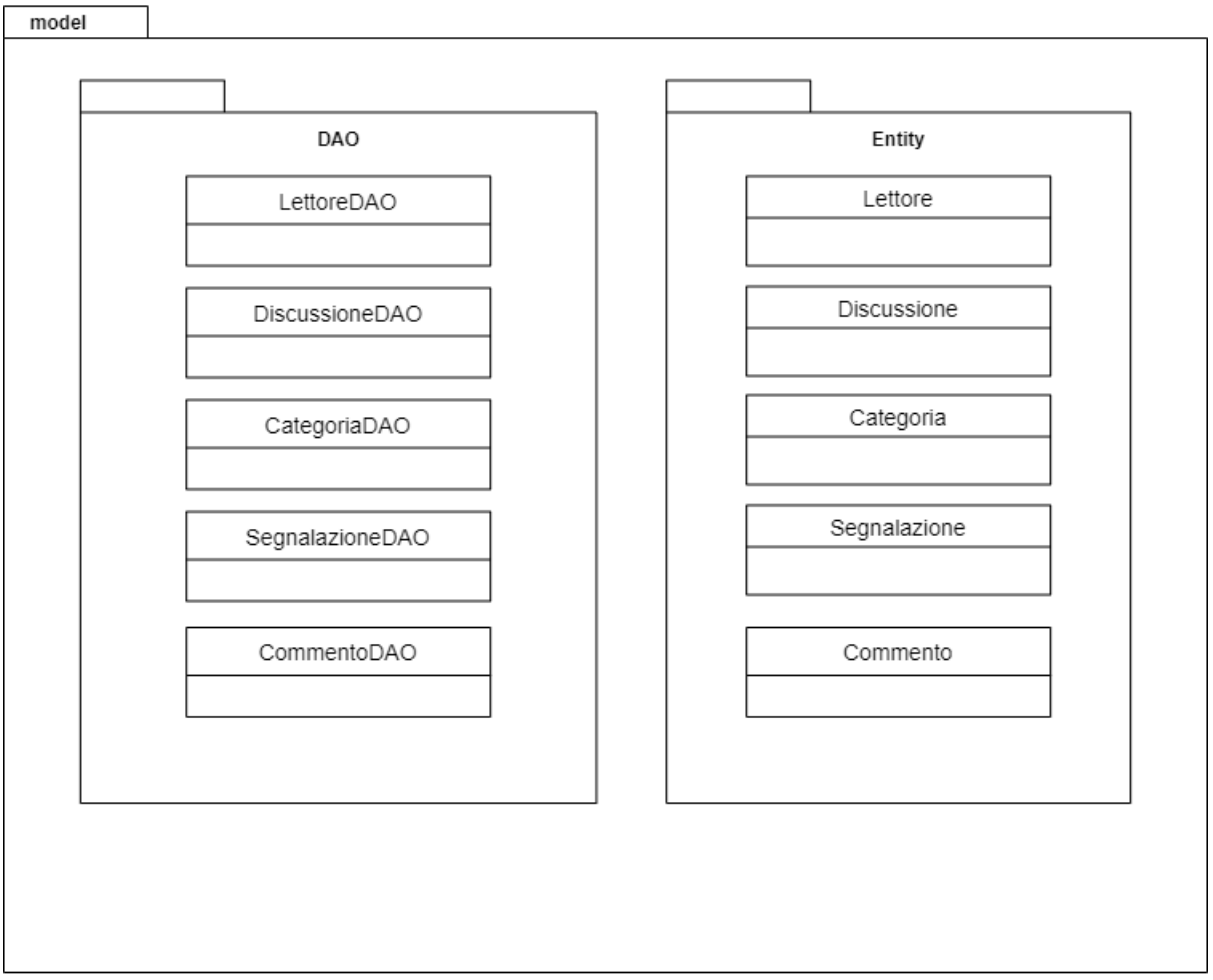
- .idea
- .mvn
- src
    - main
        - java -> contiene i package java dei sottosistemi e le relative classi che implementano
        - webapp -> contiene le pagine jsp ed html pertinenti alla view
    - test
        - java -> contiene le classi per effettuare il testing con JUnit


it.unisa.formulaOnline
autenticazione
registrazione
gestioneDiscussione
gestioneCategoria Discussione
ricerca
gestioneSegnalazione
model
dao
entity


autenticazione
service
<<interface>>
AutenticazioneService
AutenticazioneImpl
<<interface>>
GestioneUtenteService
GestioneUtenteImpl
controller
registrazione
controller
service
<<interface>>
RegistrazioneService
RegistrazioneImpl


gestioneDiscussione
controller
service
<<interface>>
DiscussioneService
DiscussioneImpl
gestioneCategoria
controller
service
<<interface>>
CategoriaService
CategoriaImpl
gestioneSegnalazione
controller
service
<<interface>>
SegnalazioneService
SegnalazioneImpl


ricerca
controller
service
<<interface>>
RicercaService
RicercaImpl


model
DAO
LettoreDAO
DiscussioneDAO
CategoriaDAO
SegnalazioneDAO
CommentoDAO
Entity
Lettore
Discussione
Categoria
Segnalazione
Commento

3. Specifica delle interfacce

Questa sezione mostra le interfacce delle classi di ogni package. In particolare sono mostrate le interfacce service gestendo la logica di business e quindi le funzioni che il sottosistema offre.
Sono omesse le funzioni di supporto (come quelle per il recupero delle entità dal database).

Nome classe	AutenticazioneService
Descrizione	Questa classe permette di gestire operazioni relative all'autenticazione di un utente
Metodi	+login(String email, String password):Lettore
Invariante	/

Nome metodo	+login(String email, String password)
Descrizione	Questa metodo permette di effettuare l'accesso tramite le credenziali
Pre-condizioni	/
Post-condizione	**context**: AutenticazioneService::login(String email, String password) **post:** **if** email e password corrispondono a Moderatore e dataFineSospensione < Date.now() **then** lettore = {oggetto corrispondente}, lettore.moderatore = True **else if** email e password corrispondono a Lettore e dataFineSospensione < Date.now() **then** lettore = {oggetto corrispondente}, lettore.moderatore = False

Nome classe	RegistrazioneService
Descrizione	Questa classe permette di gestire operazioni relative alla registrazione di un nuovo utente
Metodi	+registraLettore(String email, String password, String nickname, String scuderiaPreferita):Lettore
Invariante	/

Nome metodo	+registraLettore(String email, String password, String nickname, String scuderiaPreferita)
Descrizione	Questa metodo permette la registrazione di un lettore
Pre-condizioni	/
Post-condizione	lettore = {oggetto corrispondente} **AND** HASH(lettore.password) **AND** lettore.moderatore = False

Nome classe	LettoreService
Descrizione	Questa classe permette di gestire operazioni relative ai lettori
Metodi	+nominaModeratore(int idLettore):Lettore +aggiornaLettore(int idLettore, String email, String password, String nickname, String scuderiaPreferita, Boolean moderatore, Date dataFineSospensione):Lettore +eliminaLettore(int idLettore):void
Invariante	/

Nome metodo	+nominaModeratore(int idLettore)
Descrizione	Questa metodo permette di eleggere un moderatore
Pre-condizioni	/
Post-condizione	**context**: LettoreService::nominaModeratore(int idLettore) **post**: lettore.moderatore = True
Nome metodo	+aggiornaLettore(int idLettore, String email, String password, String nickname, String scuderiaPreferita, Boolean moderatore)
Descrizione	Modifica i dati relativi ad un lettore
Pre-condizioni	/
Post-condizione	**context**: LettoreService::aggiornaLettore(int idLettore, String email, String password, String nickname, String scuderiaPreferita, Boolean moderatore) **post:** nickname = {oggetto corrispondente}.nickname **AND** email = {oggetto corrispondente}.email **AND** password = {oggetto corrispondente}.password **AND** scuderiaPreferita = {oggetto corrispondente}.scuderiaPreferita **AND** dataFineSospensione = {oggetto corrispondente}.dataFineSospensione

Nome classe	DiscussioneService
Descrizione	Questa classe permette di gestire operazioni relative alle discussioni
Metodi	+creaDiscussione(String titolo, Int idCategoria, Int idAutore, String commento):Discussione +modificaDiscussione(String titolo, int nuovaCategoria, int idDiscussione):Discussione +eliminaDiscussione(int idDiscussione):void +aggiungiCommento(int idDiscussione, String corpo, int idAutore):Commento +modificaCommento(String corpo, int idCommento):Commento +rimuoviCommento(int idCommento, int idDiscussione):void
Invariante	/

Nome metodo	+creaDiscussione(String titolo, Int idCategoria, Int idAutore, String commento):Discussione
Descrizione	Questa metodo permette di creare una nuova discussione
Pre-condizioni	/
Post-condizione	**contex:** DiscussioneService::creaDiscussione(String titolo, Int idCategoria, Int idAutore, String commento) **post:** discussione = {oggetto corrispondente} **AND** discussione.numeroCommenti = 1
Nome metodo	+modificaDiscussione(String titolo, int nuovaCategoria, int idDiscussione)
Descrizione	Questa metodo permette di modificare i dati relativi a una discussione
Pre-condizioni	/
Post-condizione	**contex:** DiscussioneService::modificaDiscussione(String titolo, int nuovaCategoria, int idDiscussione) **post:** titolo = {oggetto corrispondente}.titolo **AND** categoria = {oggetto corrispondente}.categoria
Nome metodo	+eliminaDiscussione(int idDiscussione):void
Descrizione	Questa metodo permette di eliminare una discussione

Pre-condizioni	/
Post-condizione	/
Nome metodo	+aggiungiCommento(int idDiscussione, String corpo, int idAutore):Commento
Descrizione	Questa metodo permette di aggiungere un commento ad una discussione
Pre-condizioni	/
Post-condizione	**contex:** DiscussioneService::aggiungiCommento(int idDiscussione, String corpo, int idAutore) **post:** discussione.numeroCommenti = discussione.numeroCommenti + 1
Nome metodo	+modificaCommento(String corpo, int idCommento):Commento
Descrizione	Questa metodo permette di modificare un commento ad una discussione
Pre-condizioni	/
Post-condizione	**contex:** DiscussioneService::modificaCommento(String corpo, int idCommento) **post:** corpo = {oggettoCorrispondente}.corpo
Nome metodo	+rimuoviCommento(int idCommento, int idDiscussione)
Descrizione	Questa metodo permette di rimuovere un commento ad una discussione
Pre-condizioni	/
Post-condizione	**contex:** DiscussioneService::rimuoviCommento(int idCommento, int idDiscussione) **post:** discussione.numeroCommenti = discussione.numeroCommenti - 1

Nome classe	CategoriaService
Descrizione	Questa classe permette di gestire operazioni relative ad una categoria
Metodi	+creaCategoria(String nome, String descrizione, int categoriaPadre, int autore):Categoria +modificaCategoria(int idCategoria, String nome, String descrizione, int categoriaPadre):Categoria +eliminaCategoria(int idCategoria):void
Invariante	/

Nome metodo	+creaCategoria(String nome, String descrizione, int categoriaPadre, int autore):Categoria
Descrizione	Questa metodo permette di creare una categoria
Pre-condizioni	/
Post-condizione	**contex:** CategoriaService::creaCategoria(String nome, String descrizione, int categoriaPadre) **post:** categoria = {oggetto corrispondente}
Nome metodo	+modificaCategoria(String nome, String descrizione, int categoriaPadre):Categoria
Descrizione	Questa metodo permette di modificare le informazioni di una categoria
Pre-condizioni	/
Post-condizione	**contex:** CategoriaService::modificaCategoria(int idCategoria, String nome, String descrizione, int categoriaPadre) **post:** nome = {oggetto corrispondente}.nome AND descrizione = {oggetto corrispondente}.descrizione AND categoriaPadre = {oggetto corrispondente}.categoriaPadre
Nome metodo	+eliminaCategoria(int idCategoria):void
Descrizione	Questo metodo permette di eliminare una categoria e le sottocategorie relative. Le discussioni appartenenti alla categoria eliminata sono spostate in “senza categoria”
Pre-condizioni	/
Post-condizione	/

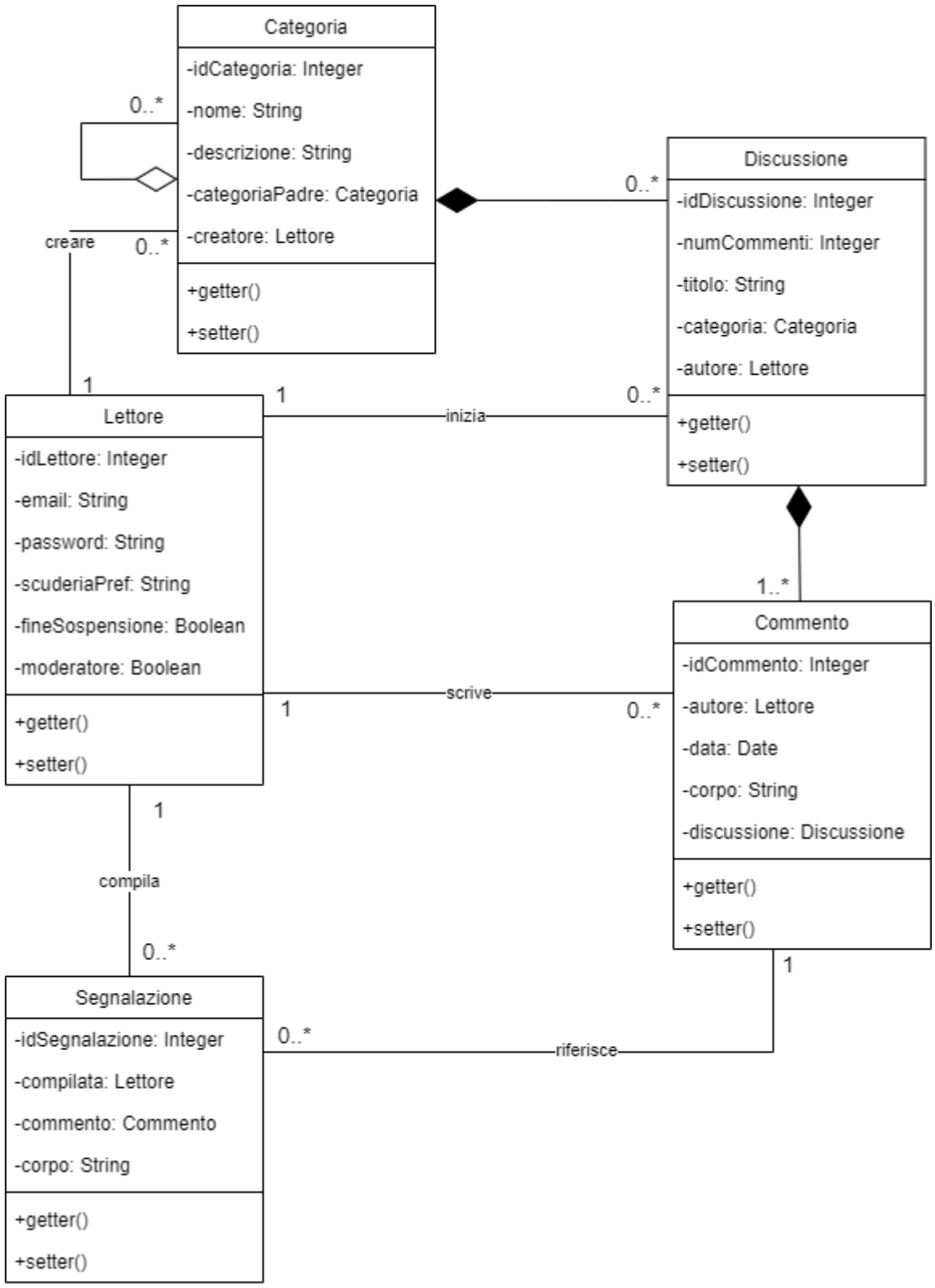
Nome classe	SegnalazioneService
Descrizione	Questa classe permette di gestire operazioni relative alle segnalazioni
Metodi	+creaSegnalazione(int idCommento, int idAutore, String corpo):Segnalazione +sospendiLettore(int idSegnalazione, Date dataFineSospensione):void +eliminaSegnalazione(int idSegnalazione):void
Invariante	/

Nome metodo	+creaSegnalazione(int idCommento, int idAutore, String corpo)
Descrizione	Questa metodo permette di segnalare un commento
Pre-condizioni	/
Post-condizione	**context**: SegnalazioneService::creaSegnalazione(int idCommento, int idAutore, String corpo) **post**: segnalazione = {oggetto corrispondente}
Nome metodo	+sospendiLettore(int idSegnalazione, Date dataFineSospensione)
Descrizione	Questo metodo permette di sospendere un lettore sospetto
Pre-condizioni	/
Post-condizione	**context**: SegnalazioneService::sospendiLettore(int idSegnalazione, Date dataFineSospensione) **post**: lettore.fineSospensione = dataFineSospensione
Nome metodo	+eliminaSegnalazione(int idSegnalazione)
Descrizione	Questo metodo permette di eliminare una segnalazione
Pre-condizioni	/
Post-condizione	/

Nome classe	RicercaService
Descrizione	Questa classe permette di effettuare ricerche di discussioni
Metodi	+ricerca(String titolo):List<Discussione>
Invariante	/

Nome metodo	+ricerca(String titolo)
Descrizione	Questo metodo permette di ricercare discussioni dato un titolo
Pre-condizioni	/
Post-condizione	/

4. Class Diagram Ristrutturato

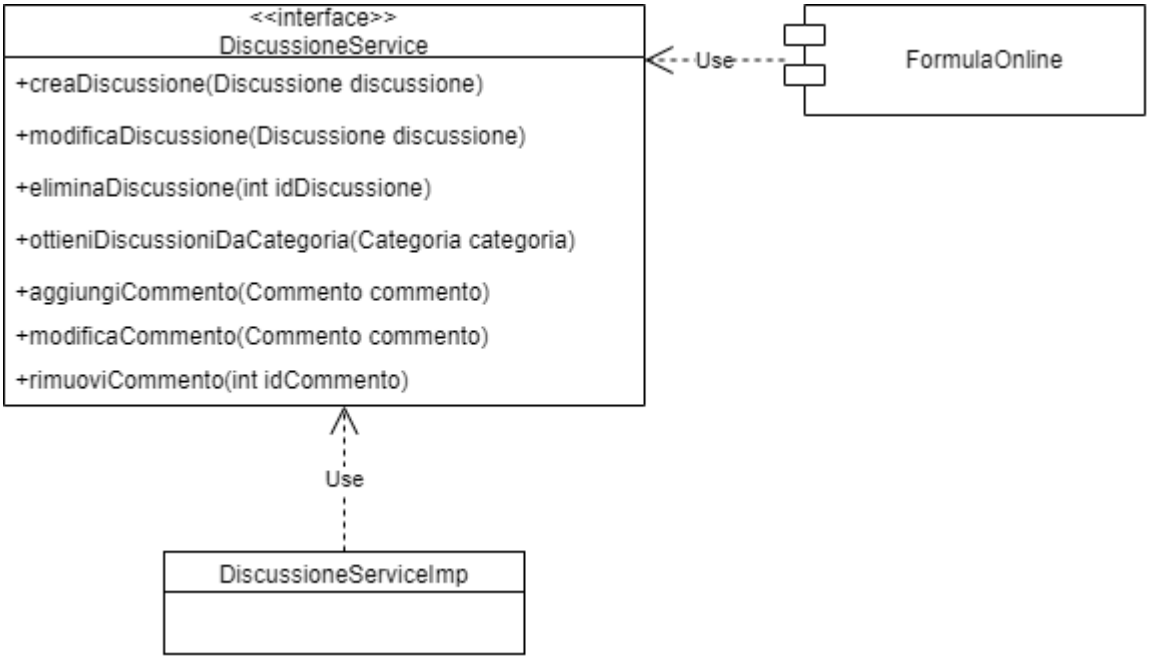


5. Elementi di riuso

5.1. Design Pattern Usati

Il design pattern Facade fornisce un' unica interfaccia per accedere ad un insieme di oggetti che compongono un sottosistema.

Definisce un'architettura chiusa.



6. Glossario

Sigla/Termine	Definizione
Categoria	Raggruppamenti di discussione secondo la tipologia
Service	Classe che implementa la logica di business relativa ad un insieme di funzionalità
Design Pattern	Template di soluzioni utilizzabili per risolvere un insieme di problemi ricorrenti. Sono composti da nome, descrizione del problema, soluzione, conseguenze.
DAO	Data Access Object, classe che permette le operazioni CRUD su un'entità in un database relazionale