Object Design Document

Progetto STORYTELLING

Riferimento	
Versione	0.3
Data	27/12/2021
Destinatario	Prof. Gravino
Presentato da	Alessandro Marigliano, Antonio Scotellaro, Emmanuele Virginio Coppola, Muriel Rossi.
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
19/12/2021	0.1	Prima stesura	Emmanuele Virginio Coppola
26/12/2021	0.2	Inserimento Packages, Interfacce delle classi, design pattern	Muriel Rossi, Alessandro Marigliano, Antonio Scotellaro, Emmanuele Virginio Coppola
27/12/2021	0.3	Inserimento design pattern, revisione	Muriel Rossi
02/01/2021	0,4	Inserimento Delle classi e i metodi	Emmanuele Virginio Coppola

Team Members

Nome	Acronimo	Informazioni di contatto
Emmanuele Virginio Coppola	EVC	e.coppola37@studenti.unisa.it
Alessandro Marigliano	AM	
Antonio Scotellaro	AS	
Muriel Rossi	MR	m.rossi67@studenti.unisa.it

Sommario

1 Introduzione

Storytelling si propone di creare una piattaforma libera di pubblicità e …

In questa prima sezione del documento verranno descritti i trade-offs e le linee guida per l'implementazione , riguardante la nomenclatura, la documentazione e le convenzioni sui formati.

1.1 Object design goals

Modularità:

Il sistema Storytelling deve avere la possibilità di poter cambiare e aggiungere funzionalità senza dover modificare molto codice.

Robustezza:

Il Sistema deve risultare robusto, reagendo in maniera adeguata alle situazioni impreviste grazie alla gestione delle eccezioni evitando di bloccare le componenti non coinvolte del sistema.

Riusabilità:

Il Sistema deve basarsi sulla riusabilità del codice attraverso l'utilizzo delle interfacce e dei design pattern

1.2 Object Trade-Off

Modularità vs Tempi di esecuzione:

Il sistema sarà costruito in maniera tale da preferire ove possibile la modularità rispetto al tempo di Esecuzione.

Robustezza vs Tempo di esecuzione:

Il sistema è costruito per preferire effettuare i controlli necessari per rendere il sistema più robusto anche se ci sono tempi di esecuzione più lughi.

1.3 Linee guida per la documentazione dell'interfaccia

Le linee guida sono una serie di regolamenti usati dagli sviluppatori per la progettazione delle interfacce per facilitarne la comprensione anche per altri team di sviluppatori. Le linee guida sono definite in base alla convenzione Java **Sun Java Coding Conventions** [Sun, 2009].

Link a documentazione ufficiale sulle convenzioni

Di seguito una lista di link alle convenzioni usate per definire le linee guida:

- **Java Sun:** https://checkstyle.sourceforge.io/sun_style.html
- **HTML:** https://www.w3schools.com/html/html5_syntax.asp

1.4 Definizioni, acronimi e abbrevizioni

Vengono riportati di seguito alcune definizioni presenti nel documento:

- **Package:** raggruppamento di classi, interfacce o file correlati;
- **Design pattern:** template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità;
- **Interfaccia:** insieme di signature delle operazioni offerte dalla classe;
- **lowerCamelCase:** è la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione nel mezzo della frase inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi;
- **UpperCamelCase:** è la pratica di scrivere frasi in modo tale che ogni parola o abbreviazione inizi con una lettera maiuscola, senza spazi o punteggiatura intermedi;
- **Javadoc:** sistema di documentazione offerto da Java, che viene generato sottoforma di interfaccia in modo da rendere la documentazione accessibile e facilmente leggibile
- **IDE(Integated develoment enviroment):** software che sfrutta l'implementazione di tecnologie diverse per velocizzare implementazione e testing del codice.
- **POJO(Plain Old Java Object):** Oggetti java semplici che non sono legati a restrizioni speciali aumentando quindi leggibilità e riutilizzo del codice.
- **CACHE:** Archivio di dati temporanei consistenti con il database

1.5 Riferimenti

Di seguito una lista di riferimenti ad altri documenti utili durante la lettura:
• Statement Of Work;
• Business Case;
• Requirements Analysis Document;
• System Design Document;
• Object Design Document;
• Test Plan;
• Matrice di tracciabilità;
• Manuale di installazione;
• Manuale utente;

Per definizioni di design pattern applicati in ambienti java:

https://www.oracle.com/java/technologies

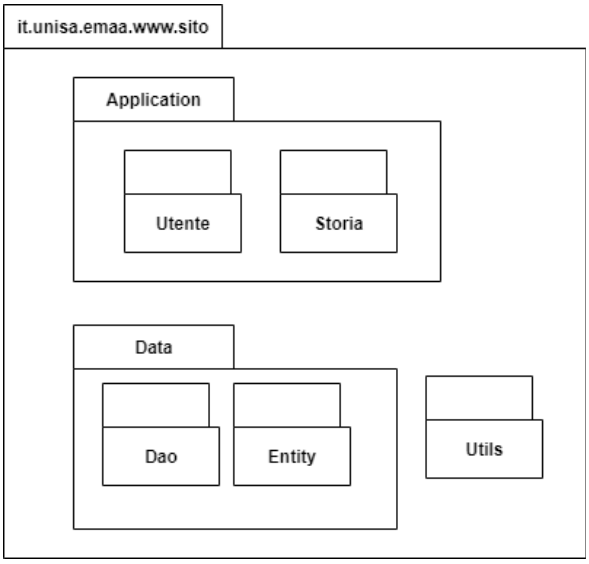
2 Packages

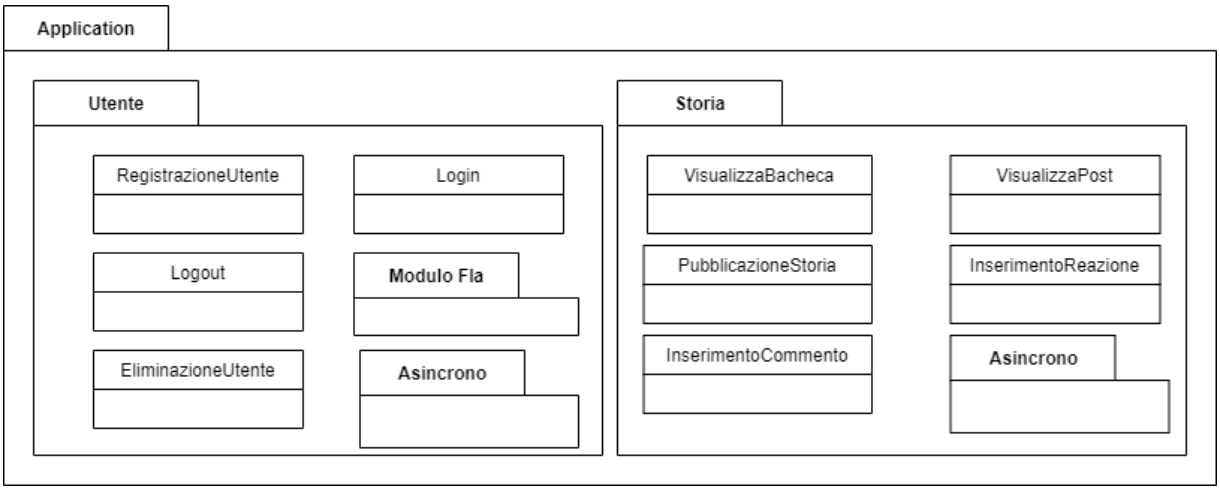
In questa sezione viene mostrata la suddivisione del sistema in package, in base a quanto definito nel documento di System Design.

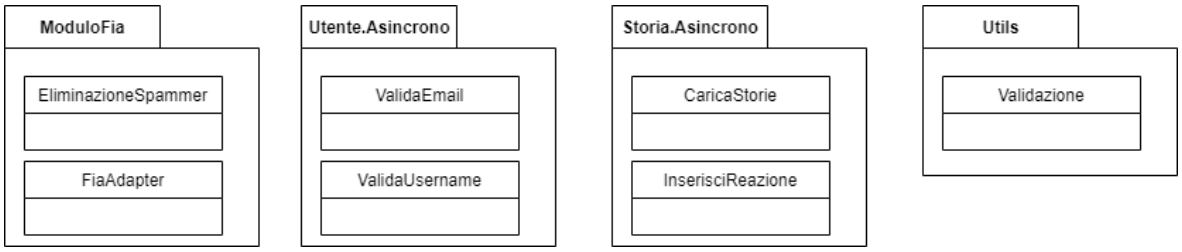
Tale suddivisione è motivata dalle scelte architetturali prese e ricalca la struttura di directory standard definita da Maven per e di quella dei progetti generati dall'IDE **Intellij IDEA** per i progetti WEB**.**

2.1Packages Java

I packages Java sono i contenitori della logica di business e della logica che gestisce la persistenza dei dati della piattaforma e quindi eseguito su nodi appositi come definiti dalla architettura three-tier, di cui i packages Java corrispondono al tier Application e Data



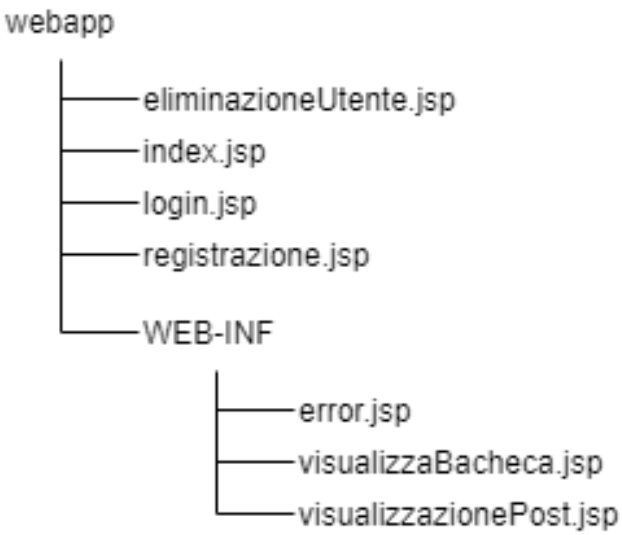




2.2 Package webapp

Il package webapp ha lo scopo di fare da interfaccia tra l'utente finale e la piattaforma usando una gui che si può far eseguire da una qualsiasi macchina che supporti un web-browser, al quale verrano inviati i contenuti del package webapp in base all'operazione richiesta.

Qui elencate si trovano le pagine jsp cioè le pagine che gestiscono il Presentation Tier della piattaforma



Classe	Descrizione
eliminazioneUtente.jsp	Pagina per l'eliminazione dell'utente.
inedx.jsp	Pagina principale per l'utente.
login.jsp	Pagina dove l'utente può effettuare il login nella piattaforma.
registrazione.jsp	Pagina dove l'utente può effettuare la registrazione nella piattaforma.
visualizzaBacheca.jsp	Pagina in cui l'utente può visualizzare le storie inserite nella piattaforma.
visualizzazionePost.jsp	Pagina in cui l'utente può visualizzare una specifica storia, compresa di commenti.

error.jsp	Pagina per la visualizzazione di eventuali errori.

3 Interfacce delle classi

Package Utils

Nome Classe Descrizione	Validazione Classe che si occupa delle operazioni di registrazione Utente e controllo della correttezza dei dati presentati
Metodi	+usernameIsPresent(String username, UtenteDao utenteDao):Boolean presente +emailRegex(String email):Boolean valido +passwordHasher(String password):String hash + reactionIsPresent(String email,int idStoria,ReazioneDao reazioneDao):Boolean presente + passwordRegex(String password):Boolean valido +datiCorrispondenti(String email,String username,String password,UtenteDao utenteDao):Boolean vero
Invarianti di Classe	/

Nome Metodo Descrizione	+usernameIsPresent(String username, UtenteDao utenteDao):Boolean presente Metodo che verifica la presenza di un'utente nel sistema
Pre-Condizione	/
Post-Condizione	/
Nome Metodo Descrizione	+emailRegex(String email):Boolean valido Metodo che verifica se la stringa passata è un'email
Pre-condizione	/
Post-Condizione	/

Nome Metodo	+passwordHasher(String password):String hash
Descrizione	Metodo che trasforma una password in chiaro in un'hash della stessa
Pre-Condizione	/
Post-Condizione	/
Nome Metodo	+ reactionIsPresent(String email,int idStoria,ReazioneDao reazioneDao):Boolean presente
Descrizione	Metodo che verifica se un'utente ha una reazione assegnata ad una determinata storia
Pre-Condizione	**context:** reactionIsPresent(String email,int idStoria,ReazioneDao reazioneDao):Boolean presente **pre:** Validazione::usernameIsPresent(String username)==true && not IStoriaDao::doRetrieveById(idStoria)==null
Post-Condizione	/
Nome Metodo	+ passwordRegex(String password):Boolean valido
Descrizione	Metodo che verifica se una password rispetta le caratteristiche di lunghezza compresa tra gli 8 e i 15 caratteri, almeno una lettere maiuscola ,almeno una lettera minuscola e almeno un numero
Pre-Condizione	/
Post-Condizione	/
Nome Metodo	+datiCorrispondenti(String email,String username,String password,UtenteDao utenteDao):Boolean vero
Descrizione	Verifica se i dati di un'utente corrispondono con quelli presenti nel database
Pre-Condizione	/

Post-Condizione	/

Package Utente

Nome Classe	RegistrazioneUtente
Descrizione	Classe che si occupa delle operazioni di registrazione Utente e controllo della correttezza dei dati presentati
Metodi	+registrazioneUtente(HttpServletRequest req,HttpServletResponse resp):void -controlloDati(String email,String password,String passwordTest,String nome,Boolean eula): Boolean corretto -effettuaRegistrazione(Utente utente):void
Invarianti di Classe	/

Nome Metodo	+registrazioneUtente(HttpServletRequest req,HttpServletResponse resp):void
Descrizione	Metodo che delega le operazioni di Registrazione Utente agli altri metodi della classe Registrazione Utente gestendone le possibili eccezioni
Pre-Condizione	/
Post-Condizione	/
Nome Metodo	-controlloDati(String email,String password,String passwordTest,String nome,Boolean eula): Boolean corretto
Descrizione	Metodo che verifica la presenza di tutti i campi richiesti e che l'email passi la relativa regular expression come anche la password
Pre-Condizione	/
Post-Condizione	/
Nome Metodo	-effettuaRegistrazione(Utente utente): Boolean corretto
Descrizione	Metodo che effettua la registrazione dell'utente

Pre-Condizione	**context:** effettuaRegistrazione(Utente utente): Boolean corretto **pre:** RegistrazioneUtente::controlloDati(String email,String password,String passwordTest,String nome,Boolean eula) && not Validazione::utenteIsPresent(Utente utente)
Post-Condizione	**context:** effettuaRegistrazione(Utente utente): Boolean corretto **post:** Validazione::utenteIsPresent(Utente utente)

Nome Classe	Login
Descrizione	Classe che gestisce il Login utente e le sue eccezioni
Metodi	+login(HttpServletRequest req,HttpServletResponse resp):void -controllaUtente(String email,String password):Utente utente -recuperaUtente(String email, String password:Boolean corretto
Invarianti di Classe	/

Nome Metodo	+login(HttpServletRequest req,HttpServletResponse resp):void
Descrizione	Metodo che delega le operazioni di Login agli altri metodi della classe Login gestendone le possibili eccezioni
Pre-Condizione	/
Post-Condizione	/
Nome Metodo	-controllaUtente(String email,String password):Boolean corretto
Descrizione	Metodo che verifica che la presenza e la corrispondenza dei dati con quelli presente nel database
Pre-Condizione	/
Post-Condizione	/

Nome Metodo	-recuperaUtente(String email):Utente utente
Descrizione	Metodo che preleva i dati dal database relativi all'utente effettuando il login e cancellando la password nella sessione
Pre-Condizione	**context:** Login:: controllaUtente(email,password) **pre:** UserDao::retrieveByEmail(email)==null
Post-Condizione	**context:** Login:: controllaUtente(email,password) **post:** isLogged(utente)==true

Nome Classe	EliminazioneUtente
Descrizione	Classe che si occupa della rimozione dell'account
Metodi	+eliminazioneUtente(HttpServletRequest req,HttpServletResponse resp):void -eliminaUtente(String email,String password):void -//da controllare il controlla Utente
Invarianti di Classe	/

Nome Metodo	+eliminazioneUtente(HttpServletRequest req,HttpServletResponse resp):void
Descrizione	Metodo che delega le operazioni di EliminazioneUtente agli altri metodi della classe EliminazioneUtente gestendone le possibili eccezioni
Pre-Condizione	/
Post-Condizione	/
Nome Metodo	-eliminaUtente(String email,String password):void
Descrizione	Metodo che elimina i dati dell'utente dal database
Pre-Condizione	**context:** Login:: controllaUtente(email,password)

	pre: not UserDao::retrieveByEmail(email)==null
Post-Condizione	**context:** Login:: controllaUtente(email,password) **post:** UserDao::retrieveByEmail(email)==null && not isLogged(utente)==true

Package Utente.Asincrono

Nome Classe	ValidaEmail
Descrizione	Classe che si occupa del controllo della presenza di un'email all'interno del database chiamando
Metodi	+validaEmail(HttpServletRequest req,HttpServletResponse resp):void
Invarianti di Classe	/

Nome Metodo	+validaEmail(HttpServletRequest req,HttpServletResponse resp):void
Descrizione	Metodo che controlla che un'email inserita sia presente o meno nel database
Pre-Condizione	/
Post-Condizione	/

Nome Classe	ValidaUsername
Descrizione	Classe che si occupa del controllo della presenza di un username all'interno del database chiamando
Metodi	+validaUsername(HttpServletRequest req,HttpServletResponse resp):void
Invarianti di Classe	/

Nome Metodo	+validaUsername(HttpServletRequest req,HttpServletResponse resp):void
Descrizione	Metodo che controlla che un username sia presente o meno nel database
Pre-Condizione	/
Post-Condizione	/

Package Utente.ModuloFIA

Nome Classe	EliminazioneSpammer
Descrizione	Classe che si occupa della cancellazione di spammer presenti sulla piattaforma
Metodi	+creaTask():void
Invarianti di Classe	/

Nome Metodo	+creaTask():void
Descrizione	Metodo che crea un timed task per l'eliminazione spammer
Pre-Condizione	/
Post-Condizione	/

Nome Classe	EliminazioneSpammer.TaskFia
Descrizione	Classe interna ad EliminazioneSpammer per essere utilizzata come classe operante del timed task ed effettuare le operazioni di comunicazione con il modulo fia
Pre-Condizione	/
Post-Condizione	/

Nome Metodo	+run():void
Descrizione	Metodo che effettua le operazioni di communicazioni con il modulo FIA
Pre-Condizione	/
Post-Condizione	/

Nome Classe	AdapterFia
Descrizione	Classe che si occupa della comunicazione tra server e modulo Fia esterno
Pre-Condizione	/
Post-Condizione	/

Nome Metodo	+utentiSpammer(ArrayList<Commento> commenti):ArrayList<String>
Descrizione	Metodo che riceve una lista di commenti , li invia a un modulo fia esterno tramite comunicazione socket e riceve una lista di utenti, che restituisce
Pre-Condizione	/
Post-Condizione	/

Package Storia

Nome Classe	VisualizzaBacheca
Descrizione	Classe che si occupa della visualizzazione della bacheca
Metodi	+visualizzaBacheca(HttpServletRequest req,HttpServletResponse resp):void
Invarianti di Classe	/

Nome Metodo	+visualizzaBacheca(HttpServletRequest req,HttpServletResponse resp):void

Descrizione	Metodo che si assicura che l'utente sia loggato in sessione e possa visualizzare la bacheca
Pre-Condizione	/
Post-Condizione	/

Nome Classe	PubblicaStoria
Descrizione	Classe che si occupa pubblicazione di una storia
Metodi	+pubblicaStoria(HttpServletRequest req, HttpServletResponse resp):void -pubblicazioneStoria(String username,String contenuto)
Invarianti di Classe	/

Nome Metodo	+pubblicaStoria(HttpServletRequest req, HttpServletResponse resp):void
Descrizione	Metodo che delega le operazioni di PubblicaStoria agli altri metodi della classe PubblicaStoria gestendone le possibili eccezioni
Pre-Condizione	**context:**utente.getUsername()==username **pre:** isLogged(utente)==true && IStoriaDao::doRetrieveAll().size==n
Post-Condizione	**context:**utente.getUsername()==username **post:** IStoriaDao::doRetrieveAll().size==n+1

Nome Classe	InserisciCommento
Descrizione	Classe che effettua l'inserimento di un commento all'interno di una storia
Metodi	+inserisciCommento(HttpServletRequest req, HttpServletResponse resp):void -inserimentoCommento(String username,int idStoria,String contenuto): boolean

Invarianti di Classe	/

Nome Metodo	+inserisciCommento(HttpServletRequest req, HttpServletResponse resp):void
Descrizione	Metodo che delega le operazioni di InserisciCommento agli altri metodi della classe InserisciCommento gestendone le possibili eccezioni
Pre-Condizione	**context:**utente.getUsername()==username **pre:** isLogged(utente)==true && IStoriaDao::doRetrieveAll().size==n
Post-Condizione	**context:**utente.getUsername()==username **post:** IStoriaDao::doRetrieveAll().size==n+1

Nome Metodo	-inserimentoCommento(String username,int idStoria,String contenuto):boolean
Descrizione	Metodo che inserisce un commento nel database
Pre-Condizione	**context:**utente.getUsername()==username && **pre:** isLogged(utente)==true && ICommentiDao::doRetrieveAll().size==n
Post-Condizione	**context:**utente.getUsername()==username **post:** ICommentiDao::doRetrieveAll().size==n+1

Nome Classe	VisualizzaPost
Descrizione	Classe che gestisce la visualizzazione di una storia e i sui commenti
Metodi	+visualizzaPost(HttpSerlvetRequest req,HttpServletResponse resp):void -recuperaPost(int idStoria,String email):Post
Invarianti di Classe	/

Nome Metodo	-visualizzaPost(HttpServletRequest req,HttpServletResponse resp):void
Descrizione	Metodo che delega le operazioni di VisualizzaPost agli altri metodi della classe VisualizzaPost gestendone le possibili eccezioni
Pre-Condizione	**context:**utente.getUsername()==username && **pre:** not IStoriaDao::doRetrieveById(int id)==null
Post-Condizione	/

Nome Metodo	-recuperaPost(int idStoria,String email):Post post
Descrizione	Metodo che recupera la storia con i suoi commenti
Pre-Condizione	**context:**utente.getUsername()==username && **pre:** not IStoriaDao::doRetrieveById(int id)==null
Post-Condizione	/

Package Storia.Asincrono

Nome Classe	InserisciReazione
Descrizione	Classe che effettua l'inserimento di una reazione a una storia
Metodi	+inserisciReazione(HttpServletRequest req,HttpServletResponse resp):void -inserimentoReazione(String email,int idStoria):boolean
Invarianti di Classe	/

Nome Metodo	+inserisciReazione(HttpServletRequest req,HttpServletResponse resp):void

Descrizione	Metodo che delega le operazioni di InserisciReazione agli altri metodi della classe InserisciReazione gestendone le possibili eccezioni
Pre-Condizione	**context:**utente.getUsername()==username && **pre:** isLogged(utente)==true && ICommentiDao::doRetrieveAll().size==n
Post-Condizione	**context:**utente.getUsername()==username **post:** ICommentiDao::doRetrieveAll().size==n+1

Nome Metodo	-inserimentoReazione(String email,int idStoria):boolean effettuato
Descrizione	Metodo che inserisce una reazione a una storia aggiungendola al database
Pre-Condizione	**context:**utente.getUsername()==username && **pre:** isLogged(utente)==true && ICommentiDao::doRetrieveAll().size==n
Post-Condizione	**context:**utente.getUsername()==username **post:** ICommentiDao::doRetrieveAll().size==n+1

Nome Classe	CaricaStorie
Descrizione	Metodo che inserisce una reazione a una storia aggiungendola al database
Metodi	+caricaStorie(HttpServletRequest req,HttpServletResponse resp):void -recuperaListaStorie(int pagina,String email):ArrayList<StoriaReazioni>
Invarianti di Classe	/

Nome Metodo	+caricaStorie(HttpServletRequest req,HttpServletResponse resp):void
Descrizione	Metodo che delega le operazioni di CaricaStorie agli altri metodi della classe CaricaStorie gestendone le possibili eccezioni
Pre-Condizione	**context:**utente.getUsername()==username && **pre:** isLogged(utente)==true && ICommentiDao::doRetrieveAll().size==n
Post-Condizione	**context:**utente.getUsername()==username **post:** ICommentiDao::doRetrieveAll().size==n+1

Nome Metodo	-recuperaListaStorie(int pagina,String email):ArrayList<StorieReazioni>
Descrizione	Metodo che recupera una lista di storie facenti parte di una pagine del database, definita da un offset e un limit
Pre-Condizione	
Post-Condizione	/

4 Design Patterns

Nella presente sezione si andranno a descrivere e dettagliare i design patterns utilizzati nello sviluppo dell'applicativo Storytelling.

Per ogni pattern si darà:
• Una brevissima introduzione teorica.
• Il problema che doveva risolvere all'interno di Storytelling.
• Una brevissima spiegazione di come si è risolto il problema in Storytelling.
• Un grafico della struttura delle classi che implementano il pattern.

Adapter

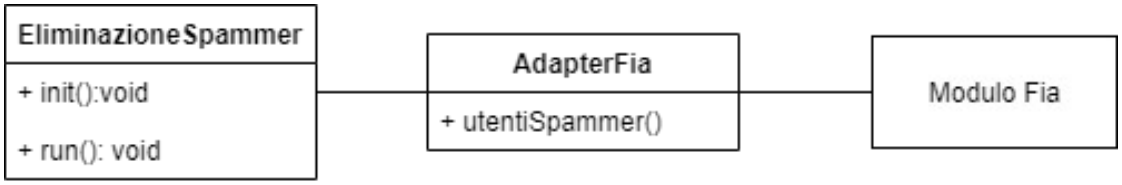
L'Adapter è un design pattern strutturale che permette oggetti con interfacce incompatibili a collaborare e/o scambiare informazioni, nascondendo la complessita di conversione.

Problema

La piattaforma deve poter communicare con il sistema esterno del modulo fia che è al di fuori del suo controllo ,di cui si ignora il meccanismo di funzionamento e il linguaggio di programmazione, ma si sa solo l'interfaccia per scambiare dati.

Soluzione

La piattaforma quando necessario comunicano con l'adapter che converte le richieste della piattaforma in messaggi comprensibili al modolo fia, in questo caso scritti in JSON, senza far trasparire tale meccanismo.



DAO

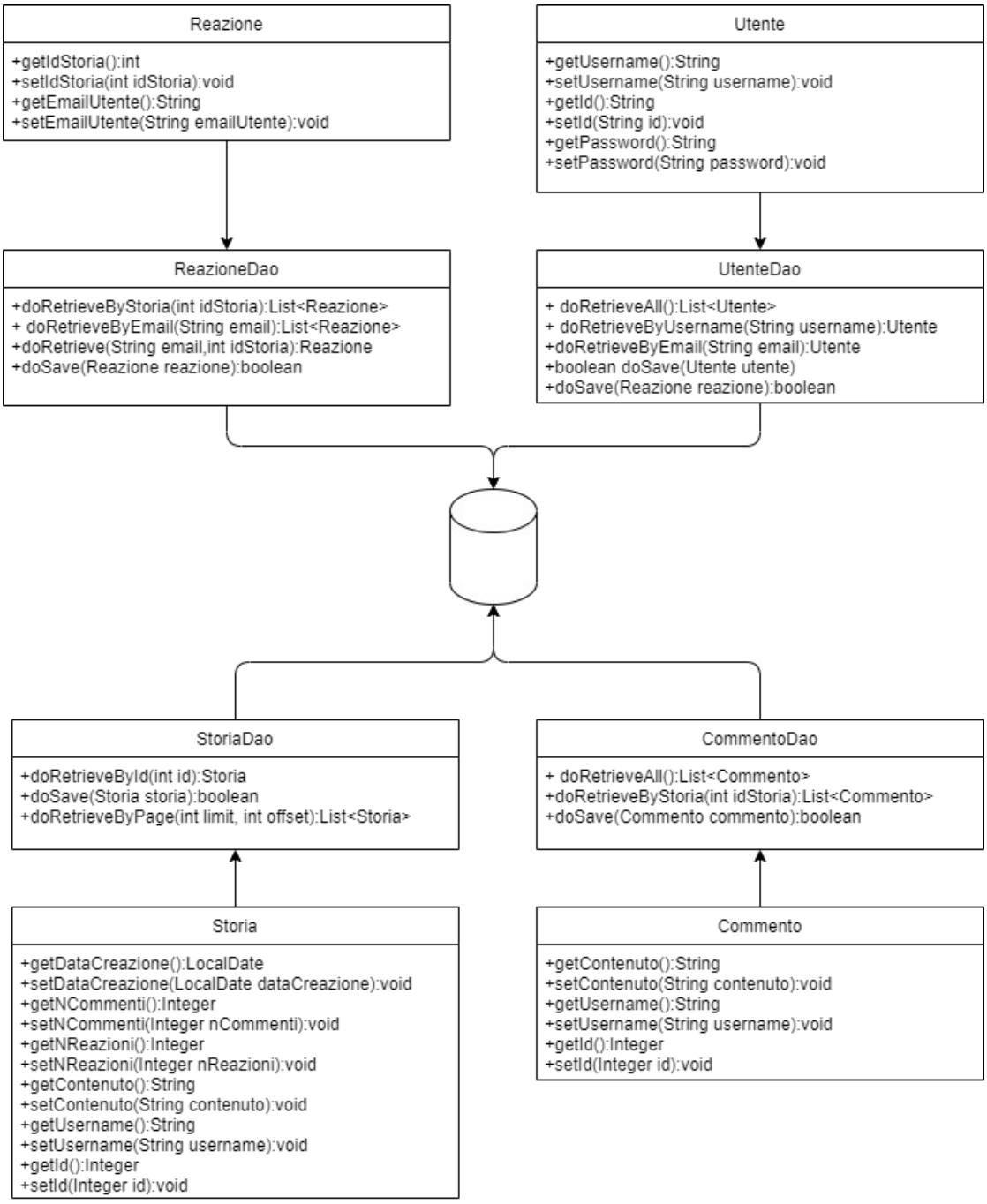
Un DAO (Data Access Object) è un pattern che offre un'interfaccia astratta, poi applicata in classi concrete ,per l'accesso dei dati ad una fonte nascondendone i dettagli implementativi agli utilizzatori e fornendogli un'oggetto POJO che contiene i dati di dati ottenuti

Problema

Rendere i dati facilmente accessibili ai vari sottosistemi senza dover applicare una logica complicata e senza dover sottoporre una quantità eccessiva di richieste al database.

Soluzione

I DAO si occuperanno di salvare e leggere i messaggi all'interno del database rendendoli maneggevoli e creando una corrispondenza tra le tabelle del database con POJO di Utenti,Commenti, Reazioni e Storie.



Singleton

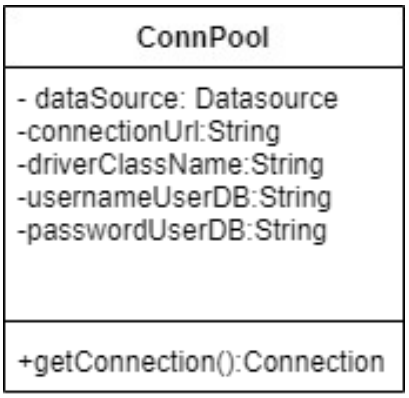
Il singleton è un pattern creazionale che permette la presenza di una sola istanza di un'oggetto alla volta e al tempo stesso renderla globalmente accessibile.

Problema

Per ottimizzare l'accesso al database bisogna gestire le varie richieste possibilmente raggruppandole ed usare soluzioni di pooling.

Soluzione

Rendere il pool di richieste unico lasciando al sistema decidere quando inoltrare le richieste in base al carico attuale e al tempo stesso non creare un numero di connessioni eccessive tra database e sistema



5 Glossario

C

Commento: Quando un'utente esprime un pensiero diretto ad una storia di un'altro utente

N

NRF: Requisito non funzionale.

R

RAD: Il documento dell'analisi dei requisiti è un'attività preliminare allo sviluppo di un Sistema software. Lo scope principale di tale documento è di definire le funzionalità del Sistema.

Reazione: Un attributo che si lasciare ad una storia per coinvolgere l'utente alla piattaforma

RF: Un requisito funzionale è un requisito che definisce una funzione del Sistema identificato durante l'analisi dei requisiti

S

Storia: Contenuto che può essere creato da un'utente a cui si possono associare commenti e reazioni

U

UC: Uno o più Use Case vengono utilizzati durante il RAD per identificare l'interazione tra attore e Sistema.