Università degli studi di Salerno
Dipartimento di Informatica
Corso di Laurea in Informatica

Ingegneria del Software

Object Design Document

“Ad Maiora”

Versione 1.0

Docente:

Andrea De Lucia

Studenti:

Nome	Matricola
Collina Ciro	0512106142
Foudal Soffian	0512105926
Meriano Davide	0512105958

Anno Accademico: 2020/2021

Data	Versione	Descrizione	Autore
21/01	0.1	Stesura indice, layout e organizzazione del documento	Davide Meriano
25/01	0.4	Definizione ed approfondimento paragrafi	Davide Meriano Soffian Foudal Ciro Collina
02/02	0.5	Modifiche layout, correzioni e ristrutturazione contenuto	Soffian Foudal
03/02	0.7	Modifiche paragrafo “Packages”	Ciro Collina
06/02	0.9	Piccole modifiche al layout, completamento del documento	Ciro Collina
07/02	1.0	Ristrutturazione “Class Interface”	Davide Meriano

1 Introduzione

Questo documento definisce l'Object Design del sistema "Ad Maiora", ossia l'aggiunta di dettagli all'analisi dei requisiti e la definizione di scelte implementative. Di seguito verranno descritte ed approfondite dinamiche già note agli altri documenti, ponendoli in funzione della prossima fase di sviluppo, ed in particolare sono trattati i trade-off di progettazione definiti dagli sviluppatori, le linee guida per la definizione di interfacce dei sottositemi, la decomposizione di quest'ultimi in pacchetti e classi, ed infine, la specifica delle interfacce di tali classi. Questo processo si inserisce in fase di progettazione come un vero punto di riferimento per gli sviluppatori, al quale s'intende offrire un'ulteriore chiave di lettura del sistema, grazie alla quale proseguire più agevolmente nelle fasi successive.

1.1 Object Design Trade-offs

- **Comprensibilità vs. Tempo**
  Il trade-off comprensibilità-tempo rappresenta una scelta ardua. Considerate approfonditamente le tempistiche in relazione alle scadenze, il tempo è una risorsa prioritaria che potrebbe guidare l'intera fase di sviluppo, tuttavia non è da sottovalutare il design del sistema, all'interno del quale viene chiaramente specificato che la comprensibilità del sistema e la sua leggibilità siano degli obiettivi di design a priorità medio-alta. La risoluzione ideale del trade-off sarebbe uno sviluppo rapido e ben organizzato, al quale affiancare degli standard medio-alti per il commento e la documentazione. Al fine di fornire una soluzione approssimata alle condizioni ideali, il codice sarà integrato da commenti ben strutturati, volti a migliorare la leggibilità, cercando di limitare il costo in termini di tempo e riducendo oculatamente lo spreco di risorse.
- **Comprare vs. Costruire**
  Questo trade-off di design rappresenta un'altra scelta importante da prendere in questa fase: optare per componenti off-the-shelf, pagandone ovviamente il prezzo, od affidarsi a componenti custom, capaci di comprendere ed incalzare il sistema al meglio. Valutando i costi, avvalendosi anche di analisi precedenti, lo sviluppo del sistema dovrebbe essere costante e non oneroso, tuttavia, lo sviluppo di componenti custom rappresenterebbe un grosso investimento in termini di tempo. La risoluzione del trade-off verte sulla gestione delle risorse, che in tal caso sembra condurre a rinunciare laddove possibile allo sviluppo di componenti custom, cercando di integrare framework ed approcci che possano aumentare la produttività e la velocità di sviluppo. La natura del sistema, interamente web-based, garantisce l'accesso ad una vastissima collezione di librerie e framework utili alla

realizzazione del prodotto, avendo un occhio di riguardo per la compatibilità.

- **Sicurezza vs. Costi**
  La risoluzione di tale trade-off sembrerebbe essere particolarmente influenzata da fattori, ossia aspetti presi in considerazione anche in fase di System Design. Seppure la sicurezza sia un aspetto fondamentale per lo sviluppo di un buon sistema, ed i costi siano già di per sé bassi, attenzioni particolari alla sicurezza del sistema potrebbero rallentare notevolmente l'implementazione. La scelta più adeguata in tal caso sarebbe quella garante dello sviluppo rapido, pertanto si intende riservare una chiara predilezione per il contenimento dei costi in termini di sviluppo.
- **Sviluppo Rapido vs. Funzionalità**
  In questa fase, dopo aver rivisto e valutato le scelte fatte durante il System Design, risulta più che opportuna la predilezione per lo sviluppo rapido, contrapposta alle funzionalità. Per questo motivo, la rispettiva implementazione seguirà un ordine decrescente di priorità, cercando di limitare le ripercussioni della scelta sulla fase di sviluppo.
- **Funzionalità vs. Usabilità**
  Il trade-off funzionalità-usabilità può considerarsi un altro pilastro del design del sistema, tratto anche da analisi precedenti sviluppate in fase di System Design. La risoluzione del trade-off vede l'usabilità come un aspetto prioritario e di grande influenza sulla progettazione, nonché prevalente. Considerando il sistema nella sua interezza, includendo le aspettative riverse nel prodotto finale, si può facilmente intuire che la rosa delle funzionalità del sistema sia limitata e controbilanciata dalle attenzioni dedicate alla loro usabilità. Il sistema è fondato sull'utente e sulle sue interazioni, ed il design del sistema cerca di comprenderlo appieno e riflettere questa caratteristica in tutte le sue fasi.

1.2 Componenti off-the-shelf

Per lo sviluppo del sistema è previsto l'utilizzo di diversi componenti off-the-shelf, ossia soluzioni "preconfezionate" messi a disposizione dal mercato, capaci di offrire strumenti capaci di velocizzare ed ottimizzare approcci e metodi orientati al problem-solving. Per la realizzazione del sistema se ne valutano i seguenti:

- **jQuery**
  È una libreria JavaScript per applicazioni web, distribuita come software libero, che nasce con l'obiettivo di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, nonché semplificare l'uso di funzionalità AJAX, la gestione degli eventi e la manipolazione dei CSS. Il framework fornisce metodi e funzioni per gestire al meglio aspetti grafici e strutturali come

posizione di elementi, effetto di click su immagini, manipolazione del Document Object Model e molto altro, mantenendo la compatibilità tra browser diversi e standardizzando gli oggetti messi a disposizione dall'interprete JavaScript del browser.

- **Bootstrap**
  Bootstrap è una raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web. Essa contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, pulsanti e navigazione, così come alcune estensioni opzionali di JavaScript. Bootstrap è compatibile con tutte le ultime versioni di tutti i principali browser e supporta il responsive web design, anche come impostazione predefinita, sottolineando il suo essere nata come libreria multi-dispositivo e multipiattaforma.
- **AJAX**
  È una tecnica di sviluppo software per la realizzazione di applicazioni web interattive, basandosi su uno scambio di dati in background fra web browser e server, consentendo l'aggiornamento dinamico di una pagina web senza l'esplicito ricaricamento della stessa da parte dell'utente. AJAX è una tecnica multi-piattaforma, utilizzabile cioè su molti sistemi operativi, architetture informatiche e browser web, e ne esistono numerose implementazioni open source di librerie e framework.
- **Jakarta Persistence (Ex JPA)**
  È un'API Jakarta EE che descrive la gestione di dati relazionali in applicazioni Java Enterprise. Nasce per risolvere il problema della persistenza dei dati ed ha unito i modelli relazionali allo sviluppo object-oriented: il percorso è composto da semplicità e robustezza, nonché ad un'ampia serie di funzionalità. L'API è utilizzabile per accedere e manipolare dati relazionali da EJB, componenti web ed applicazioni Java SE.
- **Selenium**
  È un framework portabile per il testing di applicazioni web e fornisce uno strumento di riproduzione per la creazione di test funzionali senza la necessità di utilizzare un linguaggio di scripting apposito. I test possono essere eseguiti sui più moderni web browser, nonché su diversi sistemi operativi. Selenium è un software open source rilasciato con licenza Apache, ed è composto da diverse componenti che hanno uno specifico ruolo di supporto all'automazione dei test di applicazioni web.
- **JUnit**
  È un framework per il test di unità per il linguaggio di programmazione Java, ha avuto un impatto notevole nella crescita di sviluppo Test

Driven ed è stato un modello guida per diversi framework di unit testing per altri linguaggi.

1.3 Linee guida per la documentazione delle interfacce

1.3.1 Package Model e Control

Di seguito sono elencate la struttura del progetto e le convenzioni che verranno utilizzate in fase di sviluppo per la stesura del codice:

- **Naming Conventions**
  È da considerarsi buona norma l'utilizzo di nomi: descrittivi, pronunciabili, di uso comune e frequentemente utilizzati anche in fase di documentazione, non eccessivamente lunghi od abbreviati, utilizzando solo caratteri consentiti (A-z, 0-9).
  Il formato più appropriato per i nomi è da ritenersi il Camel Case, da utilizzare nel modo più opportuno: lettere maiuscole o minuscole all'inizio della parola, rispettivamente per i nomi delle classi e per i nomi di metodi e variabili.
- **Variabili**
  I nomi delle variabili devono iniziare con lettera minuscola e proseguire in Camel Case, la loro dichiarazione deve avvenire ad inizio blocco e su righe differenti per ognuna. Eventuali dichiarazioni di variabili all'interno di metodi dovranno rispettare ugualmente tali convenzioni, che non si limitano a descrivere le variabili d'istanza.
- **Metodi**
  I nomi dei metodi devono iniziare con la lettera minuscola e proseguire in Camel Case, il loro formato deve, laddove possibile, includere un verbo, identificativo dell'azione effettuata, ed eventualmente il nome dell'oggetto su cui intenda operare o che venga prodotto dalla sua esecuzione. I blocchi sono definiti ponendo la parentesi graffa d'apertura sulla stessa riga della firma, e quella di chiusura su una riga indipendente dal corpo del metodo. Ogni metodo deve essere preceduto dal relativo Javadoc associato, di cui verrà successivamente descritta la forma.
- **Classi**
  I nomi delle classi devono iniziare con lettera maiuscola e proseguire in Camel Case, devono essere evocativi, in modo da fornire informazioni sul loro impiego all'interno del sistema. Ogni classe deve essere corredata dagli opportuni commenti e la sua definizione deve essere preceduta dal relativo Javadoc associato, di cui verrà successivamente descritta la forma.

- **Esempio riassuntivo**

```
package pacchetto;

import library;

/**
*   Breve descrizione della classe.
*
*   <pre>
*        preconditions
*   </pre>
*
*   @Author Name
*   @see    References
*/

scope class ClassName extends OtherClass implements Interfaces {
    scope type varName;

    scope ClassName(type param, ...) {...}  //Costruttore

    void setterVarName(type param) {...}    //Setter
    type getterVarName() {...}              //Getter

    /**
    *   Breve descrizione del metodo.
    *
    *   @param      descrizione parametro
    *   @return     descrizione valore di ritorno
    *   @exception  descrizione eccezione
    *   @see        References
    */
    scope return-type methodName(type param) {
        /*  Commento su
        *   più righe.
        */
        istruzioni;   //Commento su una sola riga.
        return datiDiRitorno;
    }
}
```

1.3.2 JavaScript

In fase di sviluppo verrà utilizzato JavaScript come linguaggio di scripting client-side, per aggiungere dinamicità alle pagine web. Verrà utilizzato principalmente per accedere e modificare elementi della pagina HTML, reagire ad eventi generati dall'interazione fra utente e pagina, validare i dati inseriti dall'utente ed interagire con i browser stesso. Risulta opportuno, anche in tal caso, definire le opportune convenzioni che verranno utilizzate per una corretta stesura del codice.

Gli script che svolgono funzioni distinte dal rendering dei contenuti all'interno della pagina devono essere collocati in file distinti, ed appositi. Il codice JavaScript deve seguire le stesse naming conventions definite in precedenza. I file JavaScript devono essere documentati in modo analogo a quanto descritto per il codice Java, introducendo commenti Javadoc-like laddove necessario.

1.3.3 Interfaccia Grafica

Le interfacce grafiche del sistema verranno implementate attraverso file JSP, HTML e CSS.
Le pagine JSP constano di un documento conforme allo standard HTML5 con all'interno script in codice Java che devono seguire le convenzioni definite in precedenza, aggiungendo le seguenti puntualizzazioni:
I delimitatori dei blocchi di codice java all'interno delle pagine JSP devono essere posti all'inizio di una riga, ad eccezione di codice che ne occupi una sola.
Le pagine HTML devono essere conformi allo standard HTML5 ed il codice dev'essere ben indentato.
I fogli di stile CSS si collocando in file separati e contengono tutti gli stili non in-line: le regole che lo compongono racchiudono le proprietà all'interno di parentesi graffe. Tali proprietà vengono poste una per riga, ed indentate rispetto ai selettori.

1.3.4 Database

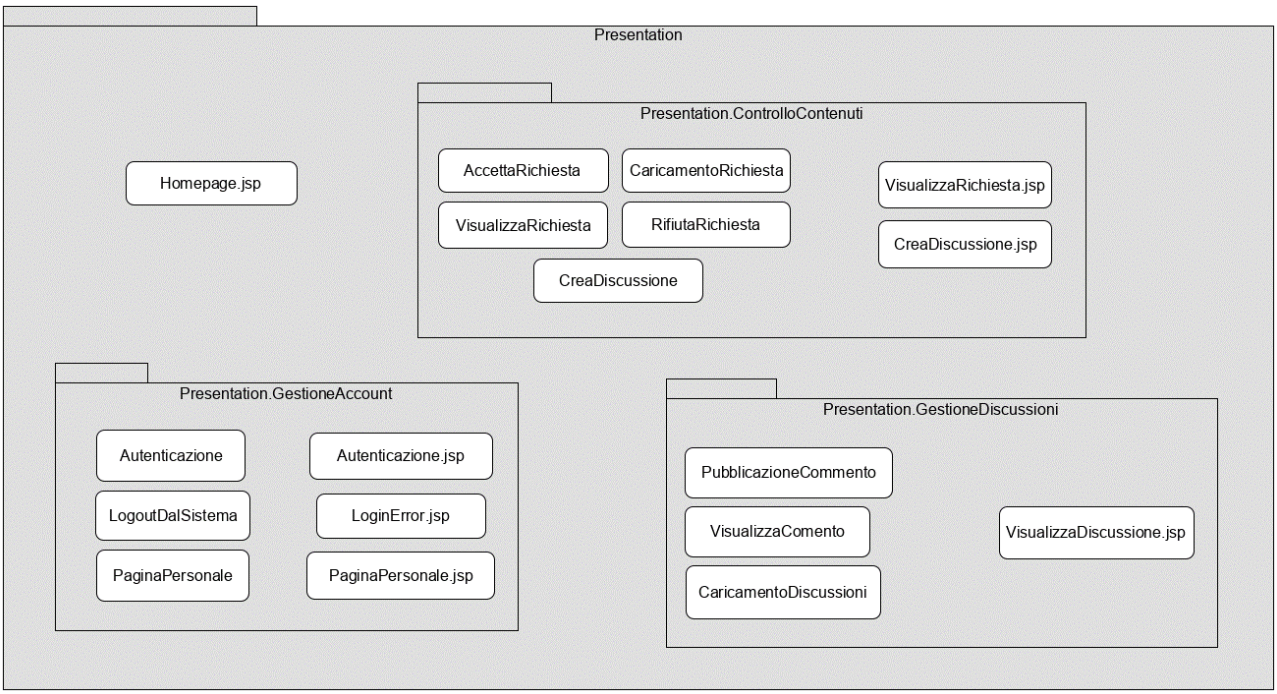
Le tabelle del Database associato al sistema devono avere la seguente forma:
I loro nomi devono essere sostantivi singolari, tratti dal dominio del problema, semplici ed esplicativi, in maiuscolo. I nomi dei loro campi devono essere espressi utilizzando le stesse convenzioni, ed è consentito l'utilizzo del carattere underscore, qualora fosse necessario l'utilizzo di più di una parola per il nome di un solo campo.

2 Definizioni, acronimi, abbreviazioni

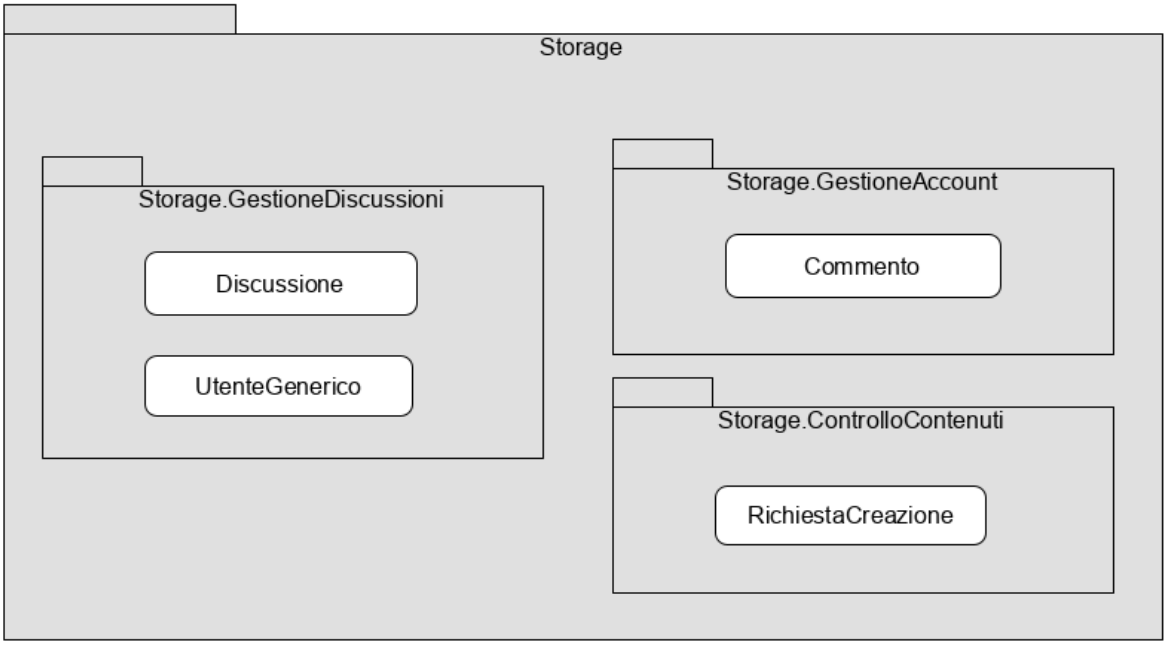
“AJAX”	:	Asynchronous JavaScript and XML
“JPA”	:	Java Persistence API, Jakarta Persistence API.
“API”	:	Application Programming Interface.
“EE”	:	Enterprise Edition – Edizione Enterprise.
“SE”	:	Standard Edition – Edizione Standard.
“pre”	:	Precondition – Precondizione.
“DOM”	:	Document Object Model – Modello ad Oggetti del Documento.
“HTML”	:	HyperText Markup Language – Linguaggio di markup per ipertesti.
“JS”	:	JavaScript.
“JSP”	:	Java Server Pages.
“CSS”	:	Cascading Style Sheets – Fogli di Stile a Cascata.
“SQL”	:	Structured Query Language.
“DAO”	:	Data Access Object.
“var”	:	Variable – Variabile.
“attr”	:	Attribute – Attributo.

3 Packages

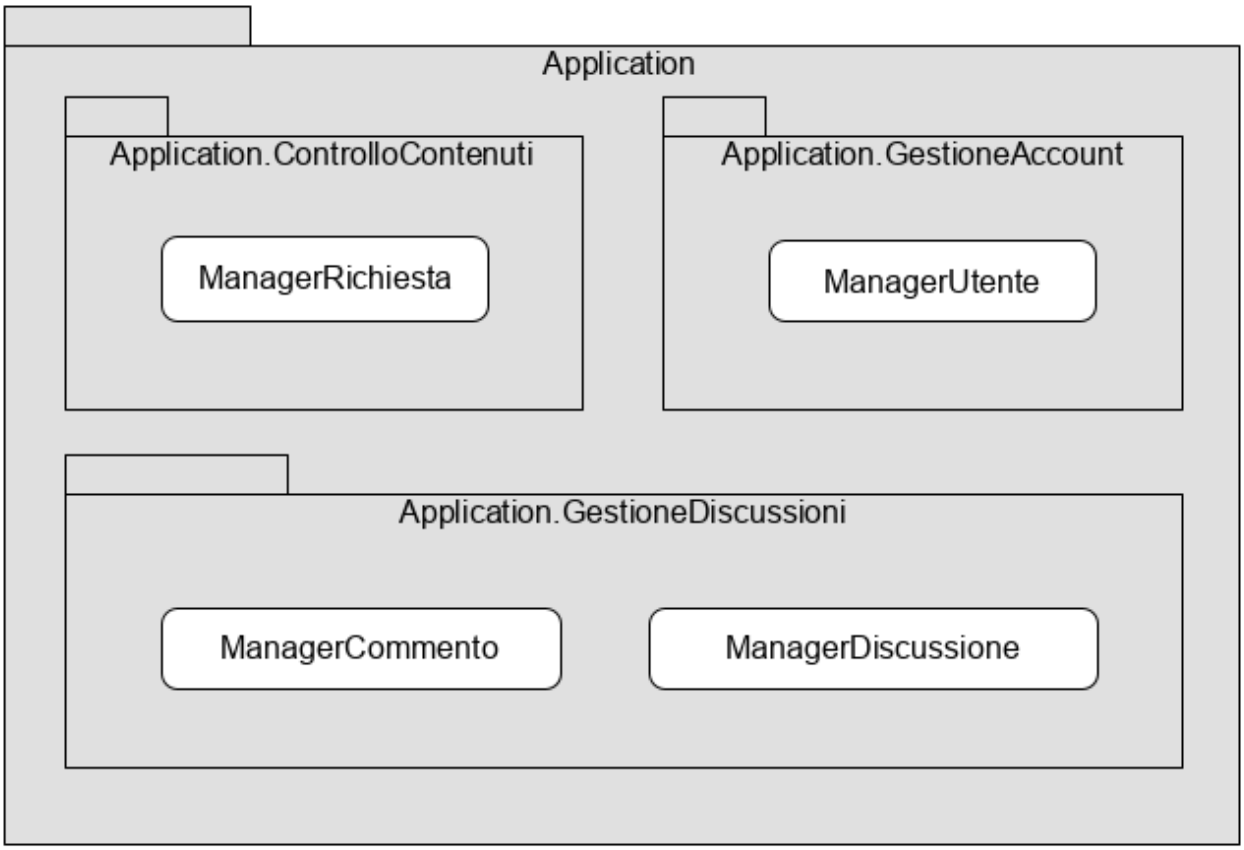
3.1 Presentation



3.2 Storage



3.3 Controller



4 Class Interface

Nome Classe :	ManagerRichiesta @ Application.ControlloContenuti

create(String, String, String, String, String, UtenteGenerico)		
context	:	ManagerRichiesta:: create(String nome, String oggetto, String descrizione, String sottocategoria, String categoria, UtenteGenerico utente): RichiestaCreazione richiesta
pre	:	nome != nil && oggetto != nil && descrizione != nil && sottocategoria != nil && categoria != nil && utente != nil
post	:	richiesta != nil
getAll()		
context	:	ManagerRichiesta :: getAll() : List richieste
pre	:	N.A.
post	:	richieste != nil **implies** richiesta.forAll(r | r != nil)
getAllPending()		
context	:	ManagerRichiesta :: getAllPending() : List richieste
pre	:	N.A.
Post	:	richieste != nil **implies** richieste.forAll(r | r != nil && r.stato==”PENDING”)
getById(String)		
context	:	ManagerRichiesta:: getById(String id): RichiestaCreazione richiesta
pre	:	id != nil
post	:	richiesta != nil **implies** richiesta.id == id
accettaRichiesta(RichiestaCreazione)		
context	:	ManagerRichiesta :: rifiutaRichiesta(RichiestaCreazione richiesta)
pre	:	richiesta != nil
post	:	@pre.stato != ”APPROVED” **implies** getById(richiesta.id).stato==”APPROVED”
rifiutaRichiesta(RichiestaCreazione)		
context	:	ManagerRichiesta :: RifiutaRichiesta(RichiestaCreazione richiesta)
pre	:	richiesta != nil
post	:	@pre.stato != ”REJECTED” **implies** getById(richiesta.id).stato==”REJECTED”

Nome Classe	**:**	ManagerUtente @ Application.ControlloContenuti

context ManagerUtente

inv: mail.includes(“@unisa.it”) **implies** utente.ruolo = “PROFESSORE”
inv: mail.includes("@studenti.unisa.it”) **implies** utente.ruolo = “STUDENTE”
inv: !mail.includes(“@unisa.it”) && !mail.includes("@studenti.unisa.it”)
implies utente.ruolo = “MODERATORE”

create(String, String, String, String, String, UtenteGenerico)		
context	**:**	ManagerUtente:: create(String username, String password, String mail, String nome, String cognome): UtenteGenerico utente
pre	**:**	username != nil && password != nil && mail != nil && nome != nil && cognome != nil
post	**:**	utente != nil
getAll()		
context	**:**	ManagerUtente :: getAll() : List utenti
pre	**:**	N.A.
post	**:**	utenti != nil **implies** utenti.forAll(r | r != nil)
getById(String)		
context	**:**	ManagerUtente::getById(String username): RichiestaUtente utente
pre	**:**	username != nil
post	**:**	utente != nil **implies** utente.username == username
login(String, String)		
context	**:**	ManagerUtente:: login(String username, String password): UtenteGenerico utente
pre	**:**	username != nil && password != nil
post	**:**	utente != nil **implies** utente.username == username && utente.password == password

Nome Classe	:	ManagerDiscussione @ Application.GestioneDiscussioni

create(String, String, String, String, String, UtenteGenerico)		
context	:	ManagerUtente:: create(String nome, String oggetto, String descrizione, String sottocategoria, String ccategoria, String mail): Discussione discussione
pre	:	nome != nil && oggetto != nil && descrizione != nil && sottocategoria != nil && categoria != nil && mail != nil
post	:	discussione != nil
getAll()		
context	:	ManagerDiscussione::getAll(): List discussioni
precondizioni	:	N.A.
post-condizioni	:	discussioni != nil **implies** discussioni.forAll(d | d != nil)
getById(String)		
context	:	ManagerDiscussione::getById(String id): Discusisone discussione
pre	:	id != nil
post	:	discussione != nil **implies** discussione.id == id

Nome Classe	:	ManagerCommento @ Application.GestioneDiscussioni

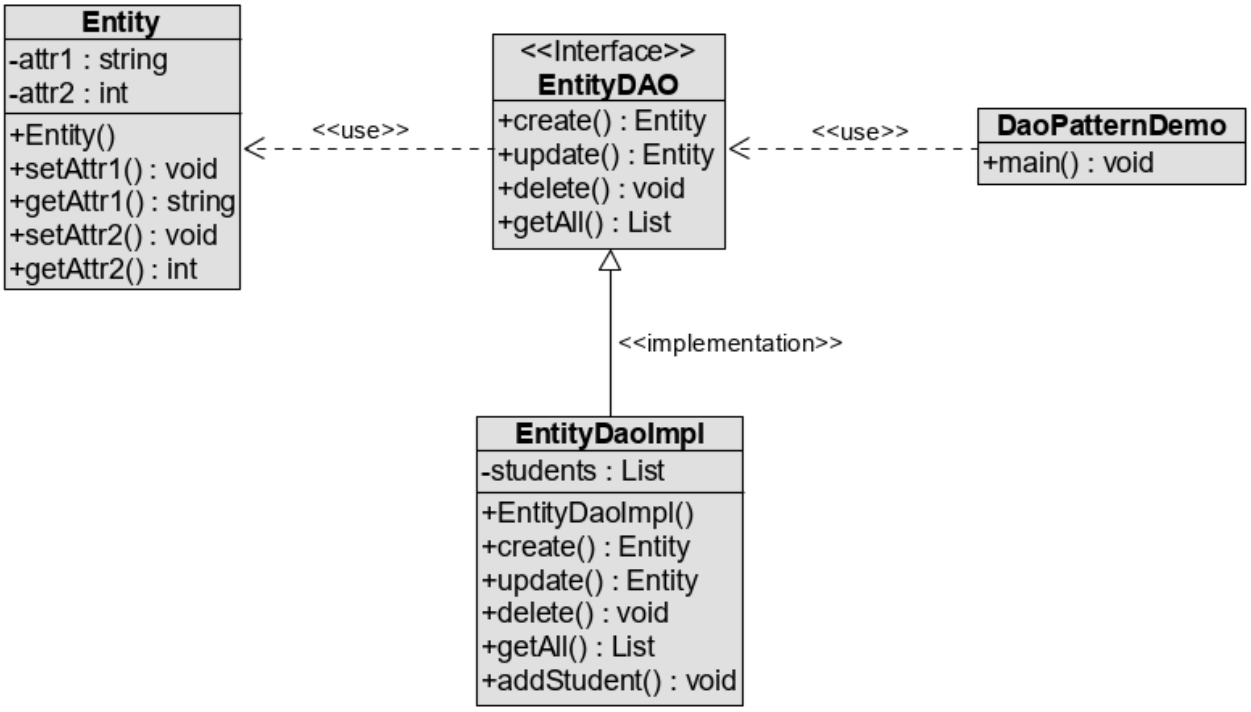
pubblica(String, UtenteGenerico, Discussione)		
context	:	ManagerUtente:: pubblica(String corpo, UtenteGenerico utente, Discussione discussione): Commento commento
pre	:	corpo != nil && utente != nil && discussione != nil
post	:	commento != nil
getAll()		
context	:	ManagerDiscussione::getAll(): List discussioni
pre	:	N.A.
post	:	discussioni != nil **implies** discussioni.forAll(d | d != nil)
getById(String)		
context	:	ManagerDiscussione::getById(String id): Discusisone discussione
pre	:	id != nil
post	:	discussione != nil **implies** discussione.id == id
getPostDiscussione(Discussione)		
context	:	ManagerCommento:: getPostDiscussione(Discussione discussione): List commenti
pre	:	discussione != nil
post	:	commenti != nil **implies** commenti.forAll(c | c.discussione == discussione)

5 Design Pattern e Class Diagram

5.1 Data Access Object Pattern

Il Data Access Object Pattern o DAO Pattern viene usato per separare le api e le operazioni di accesso ai dati dai servizi di business ad alto livello. I partecipanti al Data Object Pattern sono i seguenti:

- **Data Access Object Interface**
  Definisce le operazioni standard da effettuare sugli oggetti del model.
- **Data Access Object concrete class**
  Implementa l'interfaccia di cui sopra. È responsabile della gestione dei dati provenienti da una sorgente che può essere un database, xml, o qualsiasi altro meccanismo di storage.
- **Model Object or Value Object**
- Oggetto contenente attributi e semplici metodi getter e setter per rappresentare i dati ricevuti utilizzando la classe DAO.



6 Glossario

“Ad Maiora”	:	Il nome del gruppo di progetto, nonché quello del sistema che si intende progettare. Con questo termine, all’interno del documento, ci si riferisce unicamente al sistema. Talvolta le due parole che compongono il termine non vengono separate da spazio: ciò non altera la semantica del termine.
“Design Pattern”	:	Un design pattern descrive un problema ricorrente nell’ambiente e la sua soluzione principale, in modo che si possa riutilizzare senza cambiare approccio.
“DOM”	:	Il Document Object Model è una forma di rappresentazione dei documenti strutturati come modello orientato agli oggetti. È lo standard per la rappresentazione di documenti strutturati in maniera da essere neutrali sia per la lingua che per la piattaforma ed è nativamente supportato dai browser per modificare gli elementi di un documento HTML.
“HTML”	:	È un linguaggio di markup nato per la formattazione ed impaginazione di documenti ipertestuali disponibili nel web 1.0. Oggi è utilizzato principalmente per il disaccoppiamento della struttura logica di una pagina web e la sua rappresentazione, gestita tramite gli stili CSS per adattarsi alle nuove esigenze di comunicazione pubblicazione all’interno di internet. Talvolta lo si definisce anche attraverso il termine “HTML5” che associa al nome comune la versione a cui si fa riferimento (5 in questo caso).
“JavaScript”	:	È un linguaggio di programmazione orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati in vari modi dall’utente.
“JSP”	:	È una tecnologia di programmazione server-side che permette la creazione di metodi per la costruzione di applicazioni web-based dinamici ed indipendenti dalla piattaforma.
“CSS”	:	È un linguaggio usato per definire la formattazione di documenti HTML, XHTML ed XML, ad esempio i siti web e relative pagine. L’introduzione del CSS si è resa necessaria per separare i contenuti delle pagine HTML della loro formattazione o layout e permettere una programmazione più chiara e facile da utilizzare.
“SQL”	:	Linguaggio standardizzato per database basati sul modello relazionale, progettato per le operazioni di creazione e modifica di schemi di database, inserimento

		modifica gestione ed interrogazione dei dati memorizzati e gestione di strumenti di controllo ed accesso ai dati.
“Javadoc”	**:**	È un applicativo incluso nel Java Development Kit della Sun Microsystems, utilizzato per la generazione automatica della documentazione del codice sorgente scritto in linguaggio Java. Le informazioni di base sui package, classi, metodi e campi generate automaticamente possono essere arricchite da ulteriori dettagli per mezzo di <<commenti JavaDoc>> che vengono aggiunti alla documentazione dell’elemento che li segue.
“Responsive”	**:**	Il design responsivo, o responsive web design indica una tecnica di web design per la realizzazione di siti in grado di adattarsi graficamente in modo automatico al dispositivo coi quali vengono visualizzati, riducendo al minimo la necessità dell’utente di ridimensionare e scorrere i contenuti.