Laurea Magistrale in informatica - Università di Salerno
Corso di *Gestione dei Progetti Software* - Prof.ssa F. Ferrucci

Object Design Document

Riferimento	C17_ODD
Versione	1.0
Data	12/12/2022
Destinatario	Prof.ssa Filomena Ferrucci,
Presentato da	Michele Iannucci, Elio Testa
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
12/12/2022	0.1	Struttura del documento	Iannucci, Testa
13/12/2022	0.2	Componenti off-the-shelf	Gabriele Santoro
23/12/2022	0.3	Object design trade-offs	Michele Rabesco
23/12/2022	0.4	Design Patterns	Tutto il Team
23/12/2022	0.5	Suddivisione in packages	Biagio Andreucci
		Class Interfaces	
27/12/2022	0.6	Class Diagram	Tutto il Team
27/12/2022	0.7	Linee guida per la documentazione delle interfacce	Andrea Aceto
		Acronimi	
		Glossario	
27/12/2022	0.8	Revisione	Andrea Aceto
17/01/2023	1.0	Revisione finale	Andrea Aceto

Team

Nominativo	Ruolo
Michele Iannucci	Project Manager
Elio Testa	Project Manager
Alessandro Falcone	Team Member
Andrea Aceto	Team Member
Biagio Andreucci	Team Member
Gabriele Santoro	Team Member
Michele Rabesco	Team Member

Sommario

1. Introduzione

Cavalcando l'onda della transizione digitale, supportata dal progetto "Italia Digitale 2026", **Comun-ity** è l'idea di piattaforma Web comunale per promuovere la creazione delle Smart Communities.

1.1 Object design trade-offs

- **Sicurezza vs Tempi di risposta**: il sistema verrà implementato in modo tale da preferire l'attendibilità dei dati nel database a discapito dei tempi di risposta, che potrebbero essere maggiori.
- **Robustezza vs Tempi di risposta**: al fine di aumentare la robustezza del codice, verranno effettuati controlli sugli input, aumentando i tempi di risposta.
- **Costi di sviluppo vs Usabilità**: al fine di contenere i costi di sviluppo, l'usabilità della piattaforma sarà garantita sul 90% dei dispositivi con accesso al Web tramite i principali browser
- **Costi di sviluppo vs Portabilità**: al fine di garantire un tempo di sviluppo inferiore, il codice sarà, per la maggior parte, riusabile.
- **Costi di sviluppo vs Utilizzo di framework**: al fine di ridurre i costi di sviluppo il sistema implementerà framework e componenti già esistenti.

1.2 Componenti off-the-shelf

Il sistema comprende l'utilizzo di componenti off-the-shelf, che forniranno un supporto alla realizzazione del progetto. Tali componenti saranno:

- Bootstrap: È una raccolta di strumenti per la creazione di siti e web app, che contiene componenti per la realizzazione di interfacce basati su HTML, CSS e JS.
- jQuery: È una libreria Javascript, che facilita la manipolazione di elementi nel DOM in pagine HTML. Inoltre semplifica l'uso di funzionalità AJAX, per le funzioni di tipo asincrono.
- MongoDB: È un database non relazionale di tipo NoSQL. Differisce dai database di tipo relazionale per la sua struttura basata su documenti JSON.

1.3 Linee guida per la documentazione delle interfacce

Le linee guida per la documentazione delle interfacce contengono le convenzioni che gli sviluppatori saranno tenuti a seguire durante la progettazione e lo sviluppo delle interfacce del sistema. In particolare, per le seguenti tecnologie verranno utilizzate le indicazioni riportate nei link sottostanti:

- Java: https://checkstyle.sourceforge.io/google_style.html;
- HTML e CSS: https://google.github.io/styleguide/htmlcssguide.html;

- JavaScript: https://www.w3schools.com/js/js_conventions.asp;

1.4 Definizioni, acronimi e abbreviazioni

Definizioni: vedi Glossario.

Acronimi:

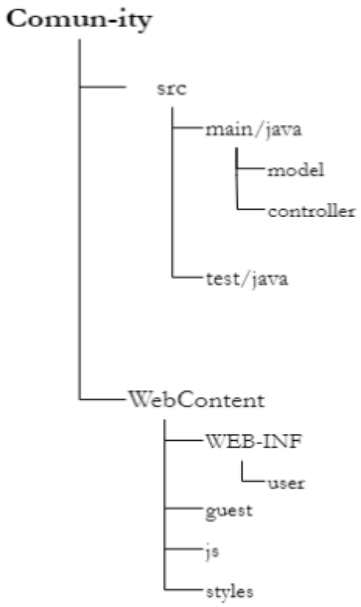
- **ODD** = Object Design Document;
- **HTML** = HyperText Markup Language;
- **CSS** = Cascading Style Sheet;
- **JS** = JavaScript;
- **AJAX** = Asynchronous JavaScript and XML;
- **XML** = eXtensible Markup Language;
- **JSON** = JavaScript Object Notation;
- **SQL** = Structured Query Language;
- **DOM** = Document Object Model;
- **DAO** = Data Access Object;
- **JSP**: Java Server Page;

1.5 Riferimenti

- B. Bruegge, A.H. Dutoit, Object Oriented Software Engineering – Using UML, Patterns and Java, Prentice Hall.
- Documento di Statement of Work relativo a questo progetto.
- Documento di Requirement Analysis Document relativo a questo progetto.
- Documento di System Design Document relativo a questo progetto.
- Linee guida per lo stile:
    - https://checkstyle.sourceforge.io/google_style.html;
    - https://google.github.io/styleguide/htmlcssguide.html;
    - https://www.w3schools.com/js/js_conventions.asp;

2. Packages

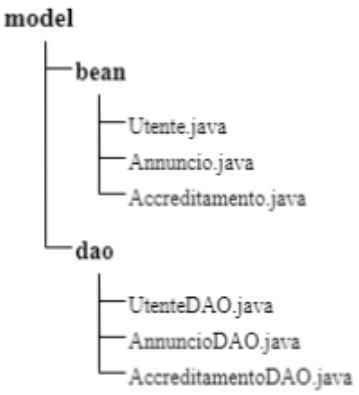
Il Sistema Proposto verrà integrato come segue:



2.1 Package Model

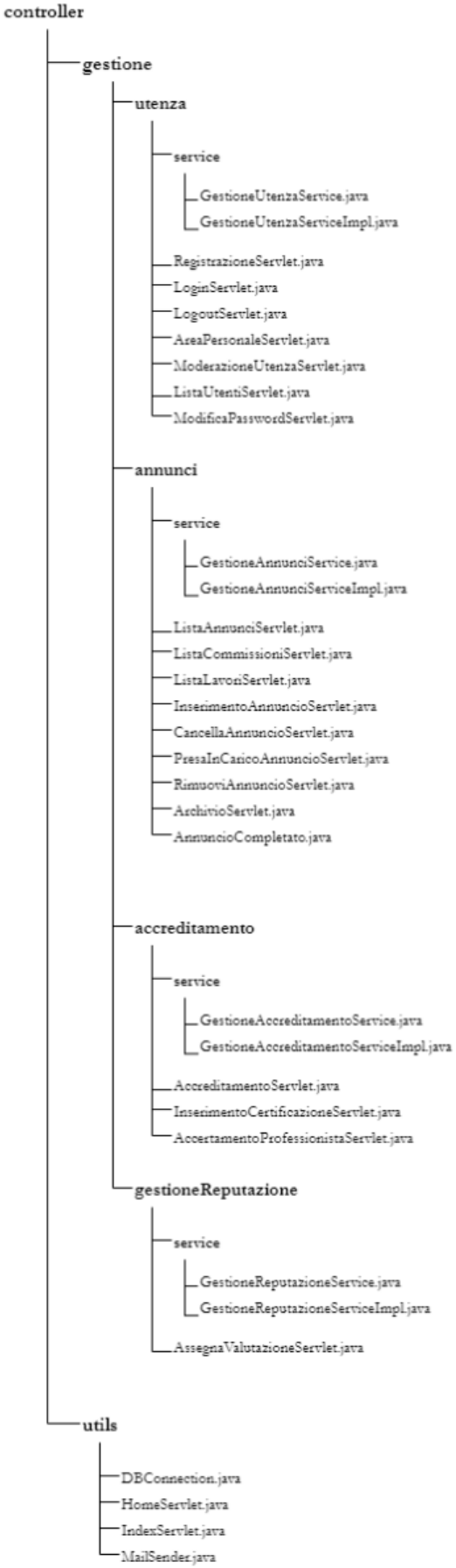
Il compito del Package "model" è di interfacciarsi dall'applicazione al Database sottostante, mediante le classi dedicate alla gestione dei dati persistenti.
All'interno di questo package troviamo due sotto-package:

- "bean", contenente le classi che rappresentano gli oggetti del nostro dominio;
- "dao", contenente i Data Access Object utilizzati per l'accesso ai dati mediante le query e in base alle operazioni da effettuare.



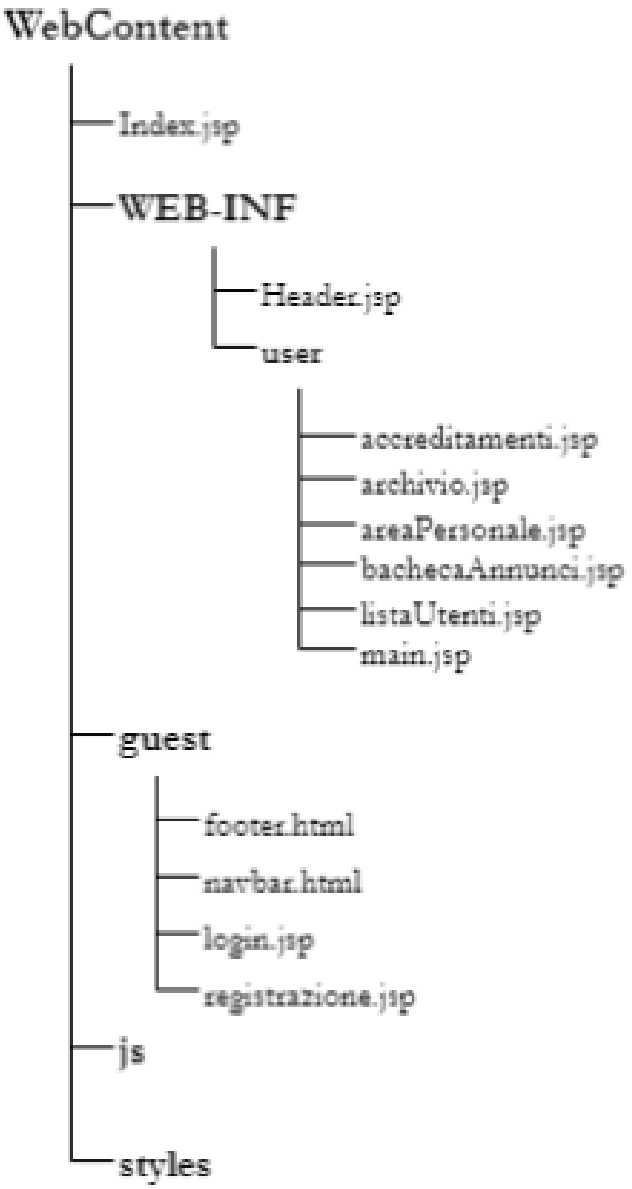
2.2 Package Controller

Il Package "controller" contiene tutte le classi dedicate alla gestione dell'interazione tra il Sistema e l'Utente, tramite il Package "WebContent", il quale funge da interfaccia fra il Client ed il Sistema. Mediante l'utilizzo delle Servlet, contenute in questo package, il Sistema potrà svolgere le funzionalità desiderate dall'Utente, al fine di semplificare le attività del lato back-end, svolte dal Sistema, ed agevolare l'Utente finale, al quale non verrà mostrata la parte Business Logic.
Il package è organizzato in cinque sotto-package, quattro dedicati alla gestione dei sottoinsiemi individuati in fase di System Design, e uno di utility contenente classi utili a tutto il Sistema.


controller
gestione
utenza
service
GestioneUtenzaService.java
GestioneUtenzaServiceImpl.java
RegistrazioneServlet.java
LoginServlet.java
LogoutServlet.java
AreaPersonaleServlet.java
ModerazioneUtenzaServlet.java
ListaUtentiServlet.java
ModificaPasswordServlet.java
annunci
service
GestioneAnnunciService.java
GestioneAnnunciServiceImpl.java
ListaAnnunciServlet.java
ListaCommissioniServlet.java
ListaLavoriServlet.java
InserimentoAnnuncioServlet.java
CancellaAnnuncioServlet.java
PresaInCaricoAnnuncioServlet.java
RinnoviAnnuncioServlet.java
ArchivioServlet.java
AnnuncioCompletato.java
accreditamento
service
GestioneAccreditamentoService.java
GestioneAccreditamentoServiceImpl.java
AccreditamentoServlet.java
InserimentoCertificazioneServlet.java
AccertamentoProfessionistaServlet.java
gestioneReputazione
service
GestioneReputazioneService.java
GestioneReputazioneServiceImpl.java
AssegnaValutazioneServlet.java
utils
DBConnection.java
HomeServlet.java
IndexServlet.java
MailSender.java

2.2 Package WebContent

Il Package "WebContent" ha lo scopo di interfacciarsi fra l'Utente finale e la parte Business Logic del Sistema. Questo Package contiene le pagine JSP, le quali fungono da interfaccia grafica all'Utente finale, al fine di utilizzare le funzionalità del Sistema in maniera semplice e veloce, non preoccupandosi di come realmente vengano effettuate le operazioni.



3. Class Interfaces

Nome classe	Utente.java
Descrizione	Questa classe rappresenta le informazioni relative all'Utente del Sistema.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome classe	Annuncio.java
Descrizione	Questa classe rappresenta le informazioni relative all'Annuncio del Sistema.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome classe	Accreditamento.java
Descrizione	Questa classe rappresenta le informazioni relative alle richieste di accreditamento.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome classe	UtenteDAO.java
Descrizione	Questa classe definisce i metodi per interrogare, inserire, aggiornare ed eliminare i dati persistenti

	dell'Utente.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome classe	AnnuncioDAO.java
Descrizione	Questa classe definisce i metodi per interrogare, inserire, aggiornare ed eliminare i dati persistenti dell'Annuncio.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome classe	AccreditamentoDAO.java
Descrizione	Questa classe definisce i metodi per interrogare, inserire, aggiornare ed eliminare i dati persistenti dell'Accreditamento.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome classe	GestioneUtenzaServiceImpl.java
Descrizione	Questa classe offre un'implementazione dell'interfaccia GestioneUtenzaService.java
Pre-condizione	-
Post-condizione	-

Invarianti	-

Nome classe	RegistarzioneServlet.java
Descrizione	Questa classe gestisce le operazioni relative alla registrazione.
Pre-condizione	**context** RegistrazioneServlet:: doPost(request:HttpServletRequest,response:HttpServletResponse) **pre**: request!=null && response!=null **context** RegistrazioneServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	**context** RegistrazioneServlet:: doPost(request:HttpServletRequest,response:HttpServletResponse) **post**: session.getAttribute("user") != null
Invarianti	-

Nome classe	LoginServlet.java
Descrizione	Questa classe gestisce le operazioni relative alla login.
Pre-condizione	**context** LoginServlet:: doPost(request:HttpServletRequest,response:HttpServletResponse) **pre**: request!=null && response!=null **context** LoginServlet:: doGet(request:HttpServletRequest,

	response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null
Post-condizione	**context** LoginServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **post**: session.setAttribute("user", user)
Invarianti	-

Nome classe	LogoutServlet.java
Descrizione	Questa classe gestisce le operazioni relative alla logout.
Pre-condizione	**context** LogoutServlet:: doPost(request:HttpServletRequest,response:HttpServletResponse) **pre**: request!=null && response!=null **context** LogoutServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null
Post-condizione	LogoutServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: session.getAttribute("user") == null
Invarianti	-

Nome classe	ModerazioneUtenzaServlet.java
Descrizione	Questa classe gestisce le operazioni relative al ban e timeout utenti.
Pre-condizione	**context** ModerazioneUtenzaServlet:: doPost(request:HttpServletRequest,response:Htt

	pServletResponse) **pre**: request!=null && response!=null **context** ModerazioneUtenzaServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	**context** ModerazioneUtenzaServlet:: doPost(request:HttpServletRequest,response:HttpServletResponse) **post**: request.setAttribute("message")
Invarianti	-

Nome classe	ListaUtentiServlet.java
Descrizione	Questa classe gestisce le operazioni relative alla visualizzazione di una lista utenti.
Pre-condizione	**context** ListaUtentiServlet:: doPost(request:HttpServletRequest,response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null && session.getAttribute("admin")!= null **context** ListaUtentiServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null && session.getAttribute("admin")!= null
Post-condizione	**context** ListaUtentiServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: reqest.setAttribute("utenti")
Invarianti	-

Nome classe	AccreditamentoServlet.java
Descrizione	Questa classe gestisce le operazioni relative all'accreditamento.
Pre-condizione	**context** AccreditamentoServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	**context** AccreditamentoServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: request.setAttribute("accreditamenti"); request.setAttribute("link");
Invarianti	-

Nome classe	AreaPersonaleServlet.java
Descrizione	Questa classe gestisce le operazioni relative all'area personale.
Pre-condizione	**context** AreaPersonaleServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null
Post-condizione	**context** AreaPersonaleServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: req.setAttribute("link"); req.setAttribute("annunci");
Invarianti	-

Nome classe	ModificaPasswordServlet.java
Descrizione	Questa classe gestisce le operazioni relative alla

	modifica della password.
Pre-condizione	**context** ModificaPasswordServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null **context** ModificaPasswordServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	**context** ModificaPasswordServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **post**: request.setAttribute("success")
Invarianti	-

Nome classe	GestioneAnnunciServiceImpl.java
Descrizione	Questa classe offre un'implementazione dell'interfaccia GestioneAnnunciService.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome classe	InserimentoAnnuncioServlet.java
Descrizione	Questa classe gestisce le operazioni relative all'inserimento di un annuncio.
Pre-condizione	**context** InserimentoAnnuncioServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null

	context InserimentoAnnuncioServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	**context** InserimentoAnnuncioServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **post**: request.setAttribute("success") **context** InserimentoAnnuncioServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: resp.sendRedirect("ListaAnnunci")
Invarianti	-

Nome classe	CancellaAnnuncioServlet.java
Descrizione	Questa classe gestisce le operazioni relative alla cancellazione di un annuncio.
Pre-condizione	**context** CancellaAnnuncioServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null **context** CancellaAnnuncioServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	**context** CancellaAnnuncioServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **post**: request.setAttribute("success") **context** CancellaAnnuncioServlet::

	doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: resp.sendRedirect("AreaPersonale")
Invarianti	-

Nome classe	ListaAnnunciServlet.java
Descrizione	Questa classe gestisce le operazioni relative alla visualizzazione di una lista annunci.
Pre-condizione	**context** ListaAnnunciServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null
Post-condizione	**context** ListaAnnunciServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **post**: request.setAttribute("link")
Invarianti	-

Nome classe	ListaLavoriServlet.java
Descrizione	Questa classe gestisce le operazioni relative alla visualizzazione di una lista di lavori.
Pre-condizione	**context** ListaLavoriServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null **context** ListaLavoriServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null

Post-condizione	**context** ListaLavoriServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **post**: response.getWriter().write(List) **context** ListaLavoriServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: response.sendRedirect("ListaAnnunci")
Invarianti	-

Nome classe	ListaCommissioniServlet.java
Descrizione	Questa classe gestisce le operazioni relative alla visualizzazione di una lista di commissioni.
Pre-condizione	**context** ListaCommissioniServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null **context** ListaLavoriServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	**context** ListaCommissioniServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **post**: response.getWriter().write(List) **context** ListaLavoriServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: response.sendRedirect("ListaAnnunci")
Invarianti	-

Nome classe	PresaInCaricoAnnuncioServlet.java
Descrizione	Questa classe gestisce le operazioni di presa in carico di una commissione/lavoro.
Pre-condizione	**context** PresaInCaricoAnnuncioServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null **context** PresaInCaricoAnnuncioServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	**context** PresaInCaricoAnnuncioServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **post**: request.setAttribute("success") **context** PresaInCaricoAnnuncioServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: response.sendRedirect()
Invarianti	-

Nome classe	ArchivioServlet.java
Descrizione	Questa classe gestisce le operazioni relative all'archiviazione di annunci.
Pre-condizione	**context** ArchivioServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null && serviceUtenza.isAdmin(user) **context**

	ArchivioServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	**context** ArchivioServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **post**: response.getWriter().write() **context** ArchivioServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: request.setAttribute("link")
Invarianti	-

Nome classe	AnnuncioCompletato.java
Descrizione	Questa classe gestisce le operazioni relative alla fine presa in carico di un annuncio.
Pre-condizione	**context** AnnuncioCompletato:: doPost(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null **context** AnnuncioCompletato:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null
Post-condizione	**context** AnnuncioCompletato:: doPost(request:HttpServletRequest, response:HttpServletResponse) **post**: request.setAttribute("success", "Annuncio marcato come completato con successo") **context** AnnuncioCompletato::

	doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: response.sendRedirect("HomeServlet")
Invarianti	-

Nome classe	GestioneReputazioneServiceImpl.java
Descrizione	Questa classe offre un'implementazione dell'interfaccia GestioneReputazioneService.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome classe	AssegnaValutazioneServlet.java
Descrizione	Questa classe gestisce le operazioni relative alla valutazione di un utente.
Pre-condizione	**context** AssegnaValutazioneServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null && serviceUtenza.isAdmin(user) **context** AssegnaValutazioneServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	**context** AssegnaValutazioneServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **post**: request.setAttribute("success") **context** AssegnaValutazioneServlet:: doGet(request:HttpServletRequest,

	response:HttpServletResponse) **post**: response.sendRedirect()
Invarianti	-

Nome classe	MailSender.java
Descrizione	Questa classe invia delle notifiche via mail.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome classe	DBConnection.java
Descrizione	Questa classe gestisce il collegamento con il database.
Pre-condizione	-
Post-condizione	**context** DBConnection:: createDBConnection(): Connection **post**: result!=null
Invarianti	-

Nome classe	IndexServlet.java
Descrizione	Questa classe invia delle notifiche via mail.
Pre-condizione	**context** IndexServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null
Post-condizione	**context** IndexServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **post**:

	response.sendRedirect()
Invarianti	-

Nome classe	HomeServlet.java
Descrizione	Questa classe invia delle notifiche via mail.
Pre-condizione	**context** HomeServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user") != null && session.getAttribute("user") != null
Post-condizione	**context** HomeServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: request.setAttribute("annunci") request.setAttribute("link")
Invarianti	-

Nome classe	InserimentoCertificazioneServlet.java
Descrizione	Questa classe gestisce le operazioni relative all'inserimento certificazione da parte del professionista.
Pre-condizione	**context** InserimentoCertificazioneServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && request.getParameter("abilitazione") && session.getAttribute("user") != null **context** InserimentoCertificazioneServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**:

	request!=null && response!=null && session.getAttribute("user") != null
Post-condizione	**context** InserimentoCertificazioneServlet:: doPost(request:HttpServletRequest, response:HttpServletResponse) **post**: request.setAttribute("success","Richiesta sottomessa con successo, verra' controllata il prima possibile") **context** InserimentoCertificazioneServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: response.sendRedirect()
Invarianti	-

Nome classe	AccertamentoProfessionistaServlet.java
Descrizione	Questa classe gestisce le operazioni relative all'accertamento di un professionista.
Pre-condizione	**context** AccertamentoProfessionistaServlet:: doPost(request:HttpServletRequest,response:HttpServletResponse) **pre**: request!=null && response!=null && request.getParameter("accettato") != null **context** AccertamentoProfessionistaServlet:: doGet(request:HttpServletRequest, response:HttpServletResponse) **pre**: request!=null && response!=null && session.getAttribute("user")
Post-condizione	**context** AccertamentoProfessionistaServlet:: doPost(request:HttpServletRequest,response:HttpServletResponse) **post**: request.setAttribute("success") **context** AccertamentoProfessionistaServlet::

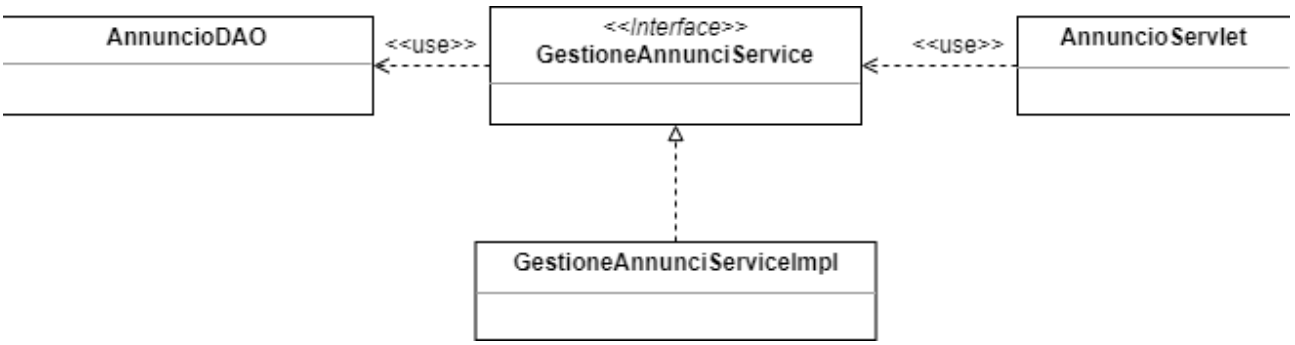
	doGet(request:HttpServletRequest, response:HttpServletResponse) **post**: response.sendRedirect()
Invarianti	-

4. Design Patterns

In questa sezione vengono illustrati i vari design pattern scelti e nello specifico come agiscono e risolvono la problematica individuata.

4.1 Facade

Il design pattern "Facade" è un pattern strutturale che definisce un'interfaccia semplificata per l'accesso ad un insieme di oggetti in un sottosistema più complesso.
Il pattern viene realizzato tramite un'interfaccia che agisce da "facciata" per fornire i metodi richiesti. Ogni sottosistema avrà, quindi, una propria interfaccia pubblica "Service", ad esempio AnnuncioService, che esporrà tutte le funzionalità fornite e realizzate dal componente Annuncio. Tale interfaccia sarà poi implementata da AnnuncioServiceImpl. Questa classe conterrà tutta la logica implementativa dei vari servizi offerti, per cui il chiamante avrà solamente bisogno di istanziarla ed utilizzare il metodo desiderato, senza dover conoscere come è stato implementato effettivamente.

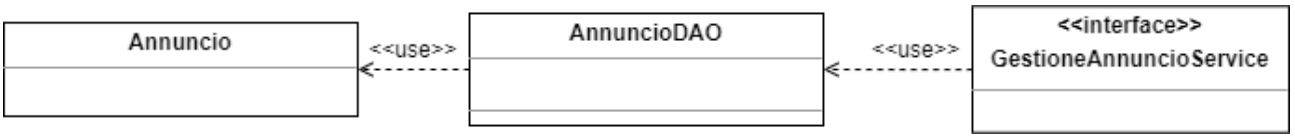


Le interfacce di alto livello espongono i vari servizi offerti da un sottosistema, rendendolo più facile da utilizzare e nascondendo dettagli implementativi o dipendenze a chi deve utilizzarlo. Le interfacce si interpongono quindi tra i vari componenti di un sistema, facilitando la comunicazione e diminuendo l'accoppiamento. Inoltre, ciò apporta vantaggi nell'effettuare l'Integration Testing.

4.2 Data Access Object

Utilizziamo questo pattern per snellire il codice dei beans e astrarre la logica di business dalla gestione della persistenza dei dati, così da non avere un alto accoppiamento tra essi.
Tramite un'interfaccia DAO si ha l'accesso al Layer di Persistenza, nascondendo all'utente i dettagli di implementazione dei metodi. Inseriamo inoltre un nuovo layer che si occupa solo della gestione dei dati persistenti, così da avere una gestione centralizzata solo per i dati persistenti.
Gli oggetti di business non essendo a conoscenza dell'implementazione del DAO garantiscono una possibile migrazione verso nuovo database, comportando modifiche solo al layer di persistenza. Si risolvono così problemi dovuti a futuri cambiamenti. Inoltre, migliora la leggibilità e la trasparenza del codice, grazie al basso accoppiamento tra i Layer.

La centralizzazione della gestione dei dati, in un livello sperato, rende più efficiente la manutenzione e la gestione dell'applicazione.



5. Class Diagram

https://drive.google.com/file/d/13yN8RIuneYgVGOJfRYMNN8xKB1pOhmVu/view?usp=sharing

6. Glossario

TERMINE	SIGNIFICATO
Object Design Document	Documento che si occupa di aggiungere dettagli all'analisi dei requisiti e prendere decisioni di implementazione.
Design Goal	Obiettivi di design progettati per il sistema proposto.
Trade-off	Scelte e compromessi tra design goals dissonanti.
Sottosistema	Un sottoinsieme dei servizi del dominio applicativo, formato da servizi legati da una relazione funzionale.
Front-end	La parte visibile all'utente di un programma e con cui egli può interagire.
Back-end	La parte che si occupa di gestire il funzionamento del sistema a seguito delle interazioni da parte del cliente nel Front-end, include anche la gestione dei dati persistenti nel Database e del sistema.
Database	Architettura esterna che si occupa della gestione e della memorizzazione dei dati persistenti.
Persistenza	Caratteristica dei dati di un programma di sopravvivere all'esecuzione del programma stesso che li ha creati, salvando i dati in uno storage non volatile
Framework	Un'architettura logica di supporto che fornisce strumenti per facilitare e velocizzare il lavoro di programmazione.
Client	Componente che accede a servizi e risorse di un altro componente detto Server.
Server	Componente che gestisce traffico di informazioni e fornisce servizi e risorse attraverso la rete.
Model	Contiene i metodi di accesso ai dati.
View	Si occupa di visualizzare i dati all'Utente e gestisce l'interazione fra quest'ultimo e l'infrastruttura sottostante.
Controller	Riceve i comandi dell'Utente attraverso il View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato del View.
Servlet	Oggetti Java all'interno del server web che permettono di creare web applications in combinazione con JSP.
Java Server Page	Tecnologia di programmazione web utilizzata per fornire contenuti dinamici.

Package	Meccanismo per organizzare classi Java in gruppi logici, principalmente allo scopo di definire spazi dei nomi distinti per diversi contesti.
Design Pattern	Una descrizione o modello logico da applicare per la risoluzione di un problema che può presentarsi in diverse situazioni durante le fasi di progettazione e sviluppo del software