

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ODD Object Design Document Choose IT – Team RocketStudios   |  |  | | --- | --- | | Riferimento |  | | Versione | 2.0 | | Data | 5/1/2019 | | Destinatario | Prof.ssa F. Ferrucci | | Presentato da | Carmine Capo, Umberto Picariello | | Approvato da |  | |

|  |
| --- |
| **Top Manager** |
| Ferrucci Filomena |

|  |  |
| --- | --- |
| **Project Manager** | |
| **Nome** | **Matricola** |
| Carmine Capo | 0522500460 |
| Umberto Picariello | 0522500485 |

|  |  |
| --- | --- |
| **Team Member** | |
| **Nome** | **Matricola** |
| Andrea Fasolino | 0512104629 |
| Igor Rinaldi | 0512104935 |
| Alessio Romano | 0512104527 |
| Mario Siglioccolo | 0512104317 |
| Vincenzo Pepe | 0512104545 |
| Marika Pia Salvato | 0515104539 |
| Maria Cosentino | 0512104929 |
| Cosimo Maio | 0512103899 |

RevisionHistory

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versione | Cambiamenti | Autori |
| 13/12/2018 | 0.1 | Prima stesura. | \* |
| 14/12/2018 | 1.0 | Revisione | Alessio Romano, Cosimo Maio |
| 31/12/2018 | 1.1 | Revisione | Igor Rinaldi |
| 05/01/2019 | 2.0 | Revisione | Cosimo Maio |

Indice

[RevisionHistory 3](#_Toc534461217)

[1. Introduzione 5](#_Toc534461218)

[1.1 Object Design Trade-offs 5](#_Toc534461219)

[1.2 Linee guida per l’implementazione 5](#_Toc534461220)

[Nomi Variabili: 6](#_Toc534461221)

[Nomi Metodi: 6](#_Toc534461222)

[Nomi Classi e JSP: 7](#_Toc534461223)

[Nomi package: 9](#_Toc534461224)

[1.3 Definizioni, acronimi e abbreviazioni 9](#_Toc534461225)

[1.4 Riferimenti 9](#_Toc534461226)

[2. Design Pattern 10](#_Toc534461227)

[2.1 Facade Design Pattern 10](#_Toc534461228)

[2.2 DAO Design Pattern 11](#_Toc534461229)

[3. Packaging 12](#_Toc534461230)

[3.1 Packages Webapp 13](#_Toc534461231)

[3.2 Packages Servlet 15](#_Toc534461232)

[3.3 Packages Services 17](#_Toc534461233)

[3.4 Packages Facade 18](#_Toc534461234)

[3.5 Packages Bean 19](#_Toc534461235)

[3.6 Packages DAO 21](#_Toc534461236)

[3.7 Packages Impl 23](#_Toc534461237)

[3.8 Packages Filters 24](#_Toc534461238)

[4. Class Interfaces 26](#_Toc534461239)

1. Introduzione

Una volta redatti i documenti relativi all’analisi dei requisiti e all’identificazione degli obiettivi di design, passiamo alla realizzazione di un ulteriore documento, l’Object Design Document.

Tale documento integrerà tutte le informazioni già evidenziate nei precedenti documenti in modo chiaro e preciso, servirà come base per l’implementazione e conterrà:

* Le specifiche legate alle interfacce per ogni sottosistema;
* Le librerie per le classi;
* I design pattern al fine di risolvere problemi e proteggere le classi da futuri cambiamenti.

1.1 Object Design Trade-offs

In questa sezione verranno definiti i trade-offs relativi al sistema “ChooseIT”:

* **Prestazioni vs. Costi**, considerando il budget a disposizione, è stato deciso di utilizzare componenti COTS, con l’obiettivo di minimizzare i costi in rapporto alla qualità delle funzionalità rilasciate;
* **Interfaccia vs. Tempo di risposta**, per garantire tempi di risposta sufficientemente brevi è stato deciso di focalizzare l’attenzione sulla realizzazione di un database performante e quanto meno complesso possibile così da permettere la realizzazione di un’interfaccia lineare e semplice;
* **Interfaccia vs Usabilità**, grazie all’utilizzo delle componenti scelte si garantisce un design gradevole e facilmente comprensibile; e conseguentemente un’interfaccia facilmente usabile;
* **Costi vs. Mantenimento**, grazie all’utilizzo delle componenti e i linguaggi d’implementazione scelti, il sistema risulterà essere facilmente manutenibile a costi ridotti;

1.2 Linee guida per l’implementazione

Le linee guide per lo sviluppo del prodotto software sono dettate da “Google Java”, ossia un documento contenente regole ben precise sullo standard da seguire durante l’implementazione.

Linee guida:

Le linee guida sono molto importanti in quanto vanno a stabilire i Naming Convention dei componenti utilizzati.

Nomi Variabili:

I nomi delle variabili devono essere brevi e significativi.

L’identificatore di una variabile è una sequenza di lettere e cifre il cui primo elemento deve essere una lettera.

I nomi delle variabili dovranno iniziare con una lettera minuscola e, qualora formati da più parole concatenate, tutte le parole successive alla prima dovranno essere capitalizzate.

(Ex: nomeVariabile).

Le variabili locali saranno dichiarate all’interno di un metodo, quindi saranno create quando il metodo verrà chiamato e cancellate al termine dell’esecuzione del metodo.

void nomeMetodo(){

int nomeVariabile=valore;

….

})

Ogni variabile sarà inizializzata prima dell’utilizzo.

Nelle variabili costanti dove il valore assunto dalle variabili non può essere modificato è possibile utilizzare “\_”.

(Ex NOME\_VARIABILE)

Le variabili di istanza saranno dichiarate esternamente alla dichiarazione di metodi.

Class NomeClasse{

int nomeVariabile=valore;

void nomeMetodo(){..}

}

Nomi Metodi:

I metodi definiscono i comportamenti degli oggetti.

I nomi delle variabili dovranno iniziare con una lettera minuscola e, qualora formati da più parole concatenate, tutte le parole successive alla prima dovranno essere capitalizzate secondo il CamelCase (“nomeMetodo()”).

Il nome di un metodo va a specificare l’azione che andrà ad eseguire quindi sarà presente almeno un verbo all’interno di esso.

(Ex. getNomeVariabileAttiva())

Ai metodi verrà aggiunto un commento JavaDoc, il quale deve essere posizionato prima della dichiarazione del metodo, e deve descriverne lo scopo. Il commento dovrà comprendere anche le informazioni riguardanti i parametri, il valore di ritorno e le eccezioni lanciate.

Nomi Classi e JSP:

I nomi delle classi dovranno iniziare con la lettera maiuscola, anche la prima lettera delle parole concatenate dovrà essere in maiuscolo.

(EX NomeClasse)

Il nome della classe deve essere intuitivo e comprensibile.

Ogni file sorgente (.class) ha al suo interno una sola classe

L’introduzione alla classe conterrà :

/\*

\*Scopo della classe

\*/

Struttura della classe:

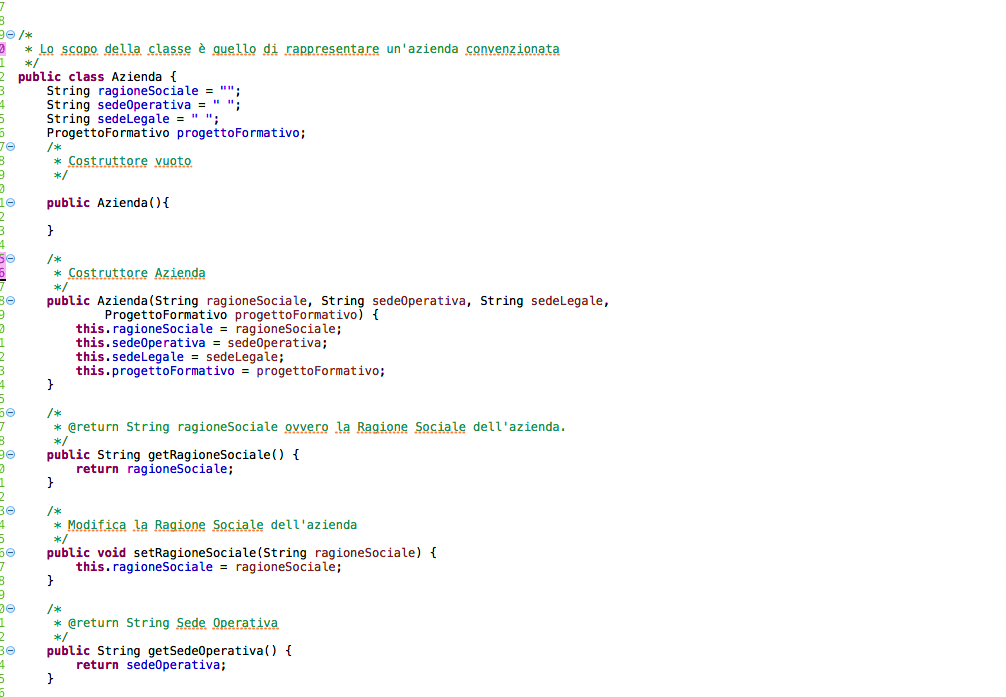
Dichiarazione classe pubblica

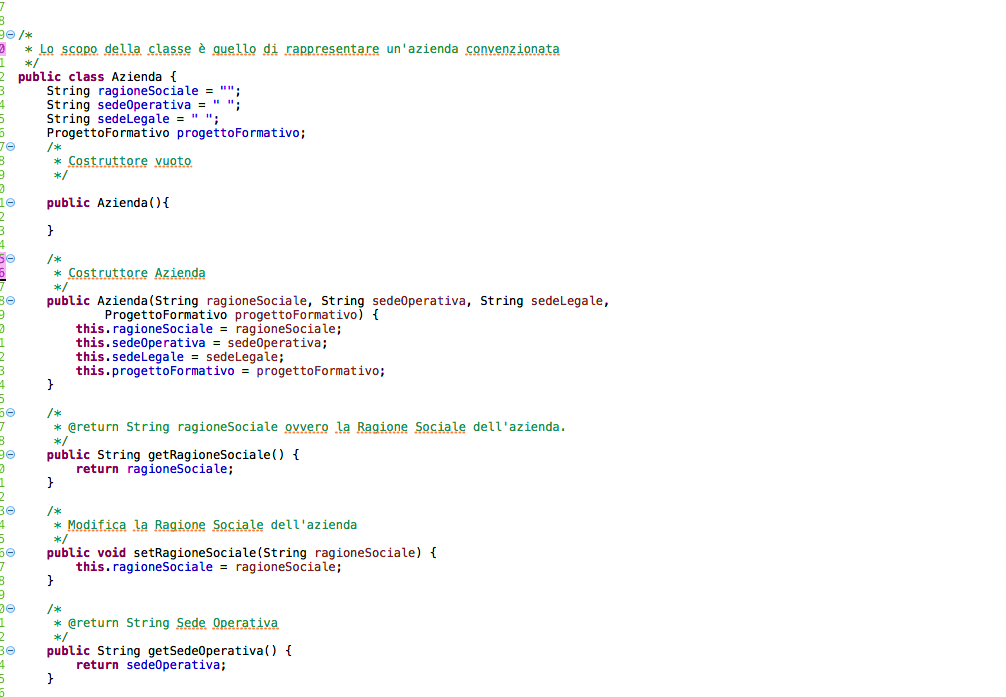
Dichiarazioni di costanti

Dichiarazioni di variali di classe e variabili d’istanza

Costruttore

Metodi e i relativi commenti





Nomi package:

I nomi dei pacchetti dovranno essere scritti in minuscolo concatenando un insieme di sostantivi separati dal punto.

In generale, un nome comincia con il [dominio di primo livello](https://it.wikipedia.org/wiki/Dominio_di_primo_livello) dell'organizzazione che lo produce, seguito dal dominio e da altri eventuali sottodomini, elencati in ordine inverso. L'organizzazione può infine scegliere un nome specifico per quel particolare package.

Non sono ammessi caratteri speciali.

1.3 Definizioni, acronimi e abbreviazioni

**Acronimi:**

RAD: Requirements Analysis Document

SDD: System Design Document

ODD: Object Design Document

DAO: Data Access Object

1.4 Riferimenti

Documento ChooseIT\_RAD\_V\_2.0.pdf

Documento ChooseIT\_SDD\_V\_3.0.pdf

**COTS (Components Off The Shelf)**

In questa sezione definiamo le Component Off The Shelf, ovvero le componenti hardware e

software disponibili sul mercato che andremo ad utilizzare nel nostro progetto software:

- JDBC: connettore (driver) per database che consente l’accesso e la gestione della

persistenza dei dati sulle basi di dati da qualsiasi programma scritto con il linguaggio di

programmazione Java, indipendentemente dal tipo di DBMS utilizzato.

- Bootstrap: raccolta di strumenti liberi per la creazione di siti e applicazioni per il Web.

Include inoltre modelli di progettazione basati su HTML e CSS, sia per la tipografia, che

per le varie componenti dell’interfaccia, come moduli, pulsanti e navigazione.

2. Design Pattern

2.1 Facade Design Pattern

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

L’applicazione di questo design pattern comporta vari vantaggi:

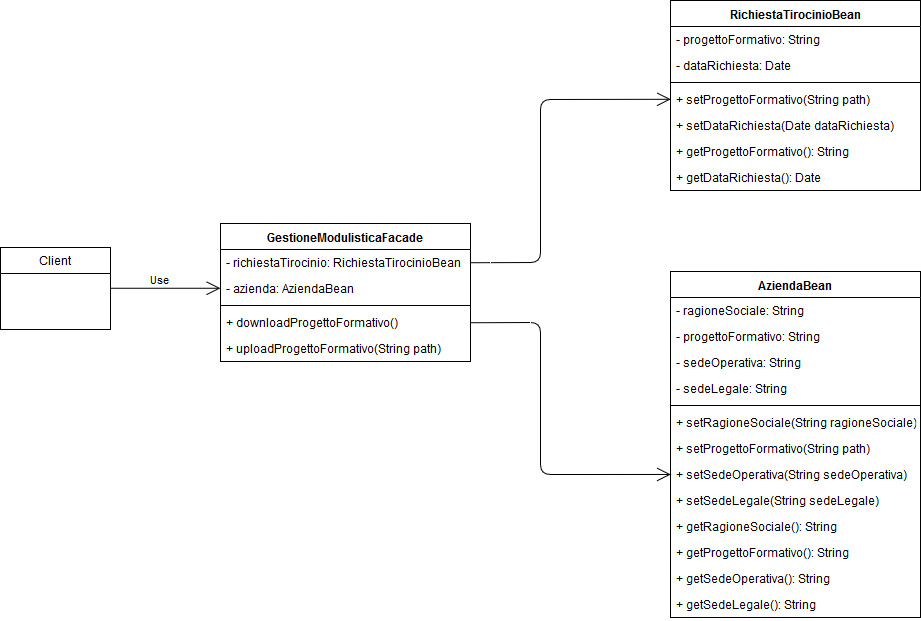
Nascondere la complessità dei servizi offerti dal sottosistema, fornendo un’architettura chiusa;

Rendere indipendenti l'implementazione di chi fa uso dei servizi del sottosistema dall'implementazione delle componenti del sistema che offrono tali servizi;

Fornire un’interfaccia unificata alle interfacce interne di un sottosistema;

Rendere il sottosistema più facile da usare, astraendone i dettagli, grazie all’uso di un'unica interfaccia;

* Promuovere un accoppiamento debole tra le componenti del sottosistema e le componenti di chi usa il sottosistema.



2.2 DAO Design Pattern

Il design pattern architetturale DAO (Data Access Object) permette di evitare una comunicazione diretta e strettamente accoppiata con il database prevedendo l’utilizzo di un’interfaccia DAO preposta a tale scopo, consentendo dunque di astrarre i dettagli implementativi dalla persistenza dei dati e favorendo manutenibilità e gestione dei dati.

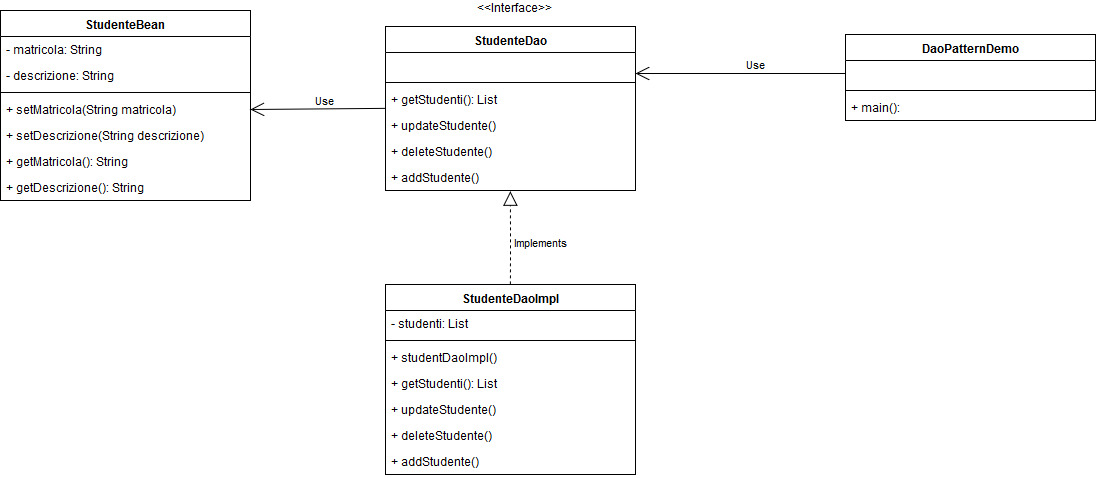
Tale design pattern ha come scopo quello di separare le operazioni di business dalle operazioni di accesso ai dati quali aggiornamento, rimozione e caricamento, che avvengono attraverso un controller che interagisce unicamente con un’interfaccia comune.

L’applicazione di questo design pattern comporta vari vantaggi:

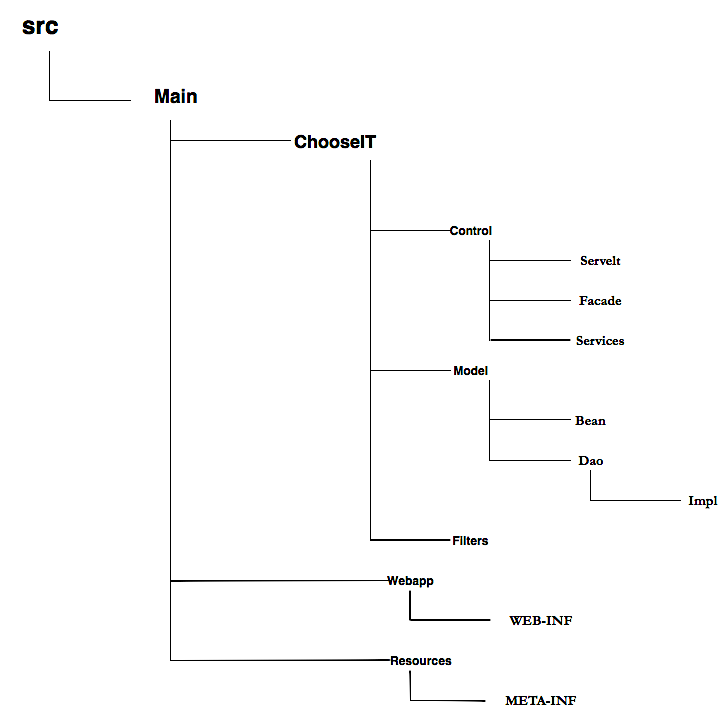
Facilità di utilizzo;

Manutenibilità;

Riutilizzo del codice.



3. Packaging



Il package “Controller” conterrà le Servlet, le Facade e i Services.

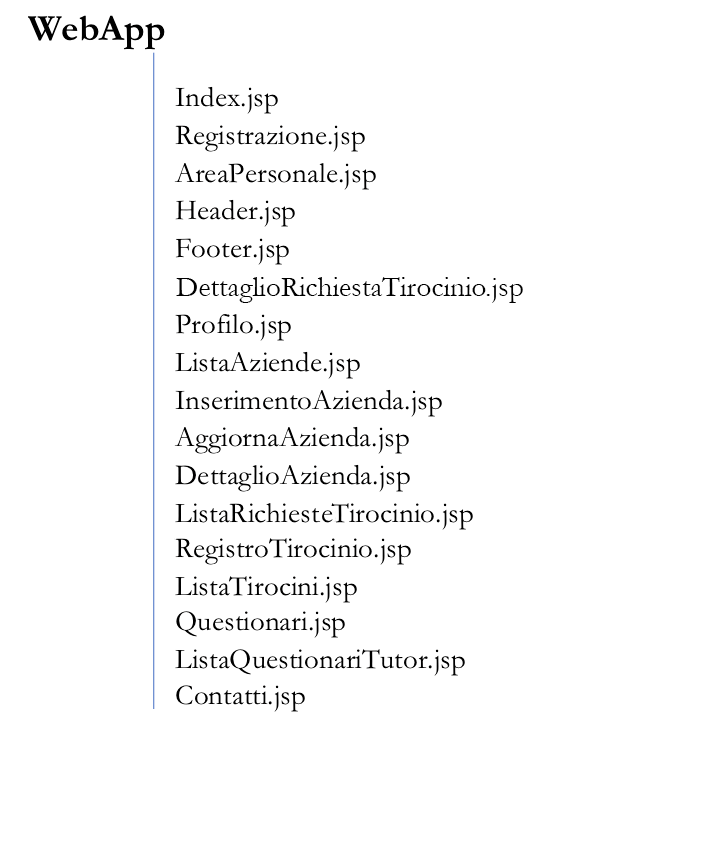
Il package “Model” conterrà Bean e Dao.

Il Package “Filters” conterrà i Filtri per il controllo degli accessi.

Il Package “Webapp” conterrà le jsp e i deployment descriptor “web.xml”

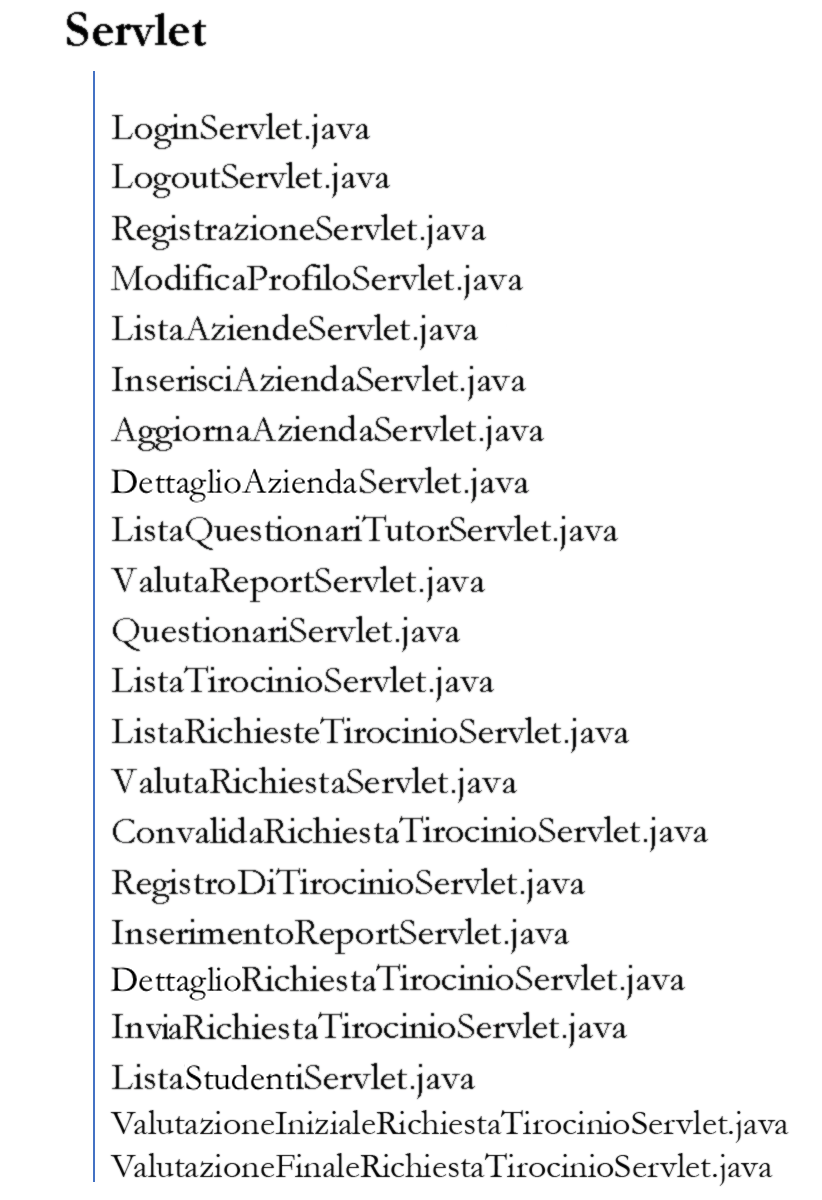
Il Package “Resources” conterrà i file per configurare.

3.1 Packages Webapp



|  |  |
| --- | --- |
| Classe | Descrizione |
| index.jsp | Pagina che permette il login o l’accesso alla pagina di registrazione al sistema. |
| Registrazione.jsp | Pagina che permette la registrazione al sistema. |
| AreaPersonale.jsp | Area personale comune a tutti gli utenti; a seconda dell’utente le operazioni che è possibile effettuare cambiano dinamicamente. |
| Header.jsp | Sezione header comune a tutte le pagine. |
| Footer.jsp | Sezione footer comune a tutte le pagine. |
| DettaglioRichiestaTirocinio.jsp | Pagina che permette di visualizzare i dettagli della richiesta di tirocinio. |
| Profilo.jsp | Pagina che permette a Studente, Tutor Universitario, Tutor Aziendale e Presidente la visualizzazione del proprio profilo. |
| ListaAziende.jsp | Pagina che permette a Studente, Tutor universitario, Segreteria e Presidente la visualizzazione della lista degli Enti convenzionati. |
| InserimentoAzienda.jsp | Pagina che permette alla Segreteria l’inserimento di una nuova Azienda. |
| AggiornaAzienda.jsp | Pagina che permette alla Segreteria la modifica di un’Azienda già presente nel sistema. |
| DettaglioAzienda.jsp | Pagina che permette la visualizzazione di tutte le informazioni su un’Azienda. |
| ListaRichiesteTirocinio.jsp | Pagina che permette a Segreteria, Presidente, Tutor Aziendale e Tutor Universitario di visualizzare le richieste di tirocinio a loro associate. |
| RegistroTirocinio.jsp | Pagina che permette a Studente, Segreteria, Tutor Aziendale, Tutor Universitario e Presidente di visualizzare un Registro di Tirocinio. |
| ListaTirocini.jsp | Pagina che permette a Segreteria, Presidente , Tutor Aziendale e Tutor Universitario di visualizzare la lista di tirocini a loro associati. |
| ListaQuestionariTutor.jsp | Pagina che permette al Tutor Aziendale di visualizzare la lista dei questionari valutazione Studente relativi agli studenti a lui associati. |
| Questionari.jsp | Pagina che permette allo Studente di visualizzare e compilare il Questionario Valutazione Ente Ospitante e al Tutor Aziendale di visualizzare e compilare il Questionario Valutazione Studente. |
| Contatti.jsp | Pagina che permette la visualizzazione dei contatti del dipartimento di Informatica. |

3.2 Packages Servlet



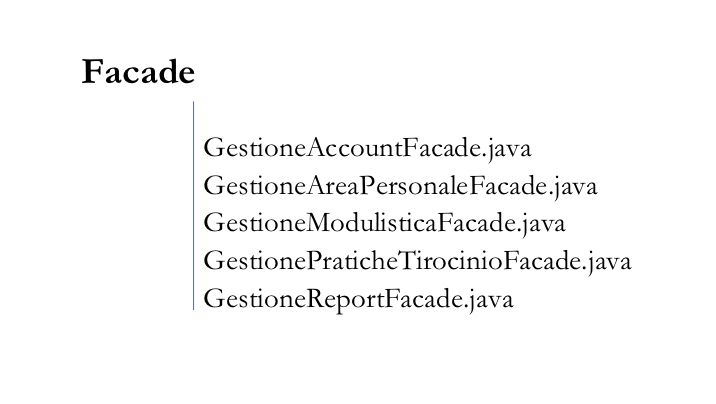
|  |  |
| --- | --- |
| Classe | Descrizione |
| LoginServlet.java | Servlet che gestisce il login al sistema. |
| LogoutServlet.java | Servlet che gestisce il logout dal sistema. |
| RegistrazioneServlet.java | Servlet che gestisce la registrazione al sistema. |
| ModificaProfiloServlet.java | Servlet che gestisce la modifica di un profilo. |
| ListaAziendeServlet.java | Servlet che gestisce il recupero della lista delle Aziende convenzionate. |
| InserisciAziendaServlet.java | Servlet che gestisce l’inserimento di una nuova Azienda da parte della Segreteria. |
| AggiornaAziendaServlet.java | Servlet che gestisce la modifica di un’Azienda da parte della Segreteria. |
| ValutazioneInizialeRichiestaTirocinioServlet.java | Servlet che gestisce la valutazione iniziale della richiesta di tirocinio. |
| ListaStudentiServlet.java | Servlet che gestisce il recupero della lista degli studenti. |
| ListaRichiesteTirocinioServlet.java | Servlet che gestisce il recupero della lista di richieste di tirocinio pendenti. |
| ValutazioneFinaleRichiestaTirocinioServlet.java | Servlet che gestisce la valutazione finale di una richiesta di tirocinio da parte della Segreteria. |
| ConvalidaRichiestaTirocinioServlet.java | Servlet che gestisce la convalida di una richiesta di tirocinio da parte del Presidente. |
| ListaTirocinioServlet.java | Servlet che gestisce il recupero della lista di tirocini presenti nel sistema. |
| DettaglioAziendaServlet.java | Serlvet che gestisce i dettagli relativi ad un'azienda. |
| DettaglioRichiestaTirocinioServlet.java | Servlet che gestisce i dettagli di una richiesta di tirocinio. |
| InviaRichiestaTirocinioServlet.java | Servlet che gestisce l’invio di una richiesta di tirocinio. |
| InserimentoReportServlet.java | Servlet che gestisce l’aggiunta di un nuovo report al registro di tirocinio da parte dello Studente. |
| ValutaReportServlet.java | Servlet che gestisce la valutazione di un Report di uno studente da parte del Tutor Aziendale. |
| listaQuestionariTutorServlet.java | Servlet che gestisce il recupero della lista di questionari valutazione Studente in attesa di compilazione da parte di un Tutor Aziendale |
| questionariServlet.java | Servlet che gestisce la compilazione di un questionario valutazione Ente Ospitante o di un questionario valutazione Studente. |

3.3 Packages Services



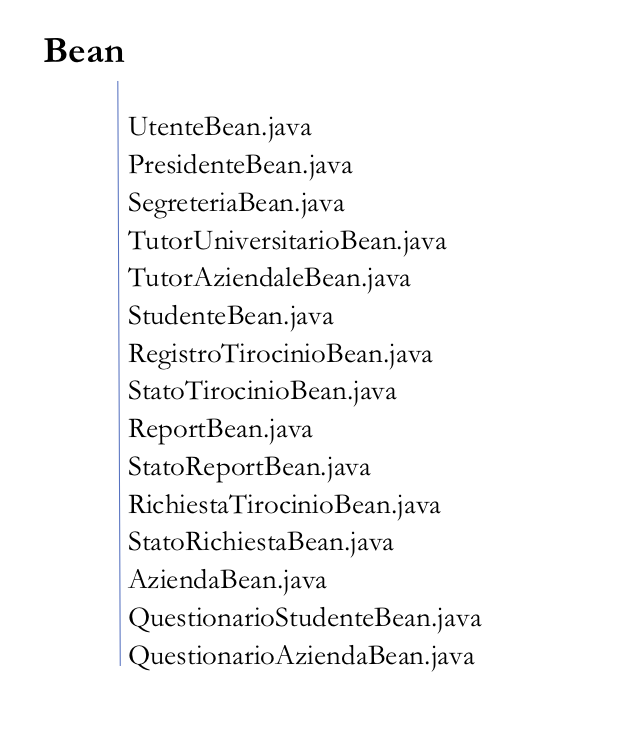
|  |  |
| --- | --- |
| Classe | Descrizione |
| DriverManagerCnnectionPool.java | Classe che gestisce la creazione di una serie di connessioni al database e il mantenimento delle stesse per la realizzazione di una connection pool. |
| ConvertEnum.java | Classe che converte le stringhe, indipendentemente dal loro formato, nella corrispettiva variabile enum quando si ha a che fare con gli stati delle richieste di tirocinio, dei report e dei tirocini. |
| GestorePassword.java | Classe che gestisce le password degli utenti. |
| ReportKey.java | Classe che rappresenta la chiave primaria per i report |

3.4 Packages Facade



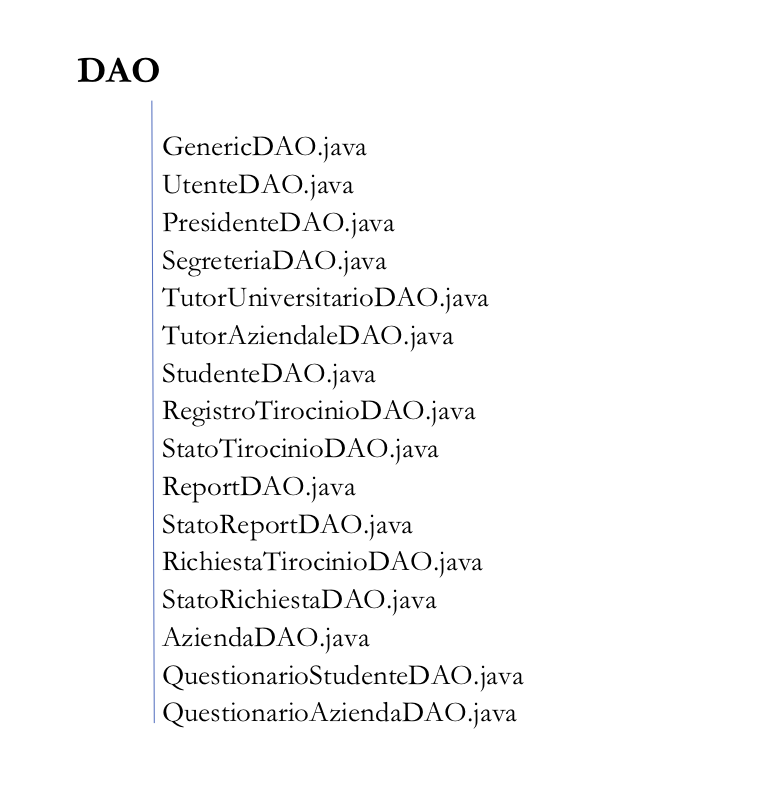
|  |  |
| --- | --- |
| Classe | Descrizione |
| GestioneAccountFacade.java | Classe Facade che funge da interfaccia tra la view (e le servlet) e tutte le operazioni del sottosistema Gestione Account. |
| GestioneAreaPersonaleFacade.java | Classe Facade che funge da interfaccia tra la view (e le servlet) e tutte le operazioni del sottosistema Gestione Area Personale. |
| GestioneModulisticaFacade.java | Classe Facade che funge da interfaccia tra la view (e le servlet) e tutte le operazioni del sottosistema Gestione Modulistica. |
| GestionePraticheTirocinioFacade.java | Classe Facade che funge da interfaccia tra la view (e le servlet) e tutte le operazioni del sottosistema Gestione Pratiche Tirocinio. |
| GestioneReportFacade.java | Classe Facade che funge da interfaccia tra la view (e le servlet) e tutte le operazioni del sottosistema Gestione Report. |

3.5 Packages Bean



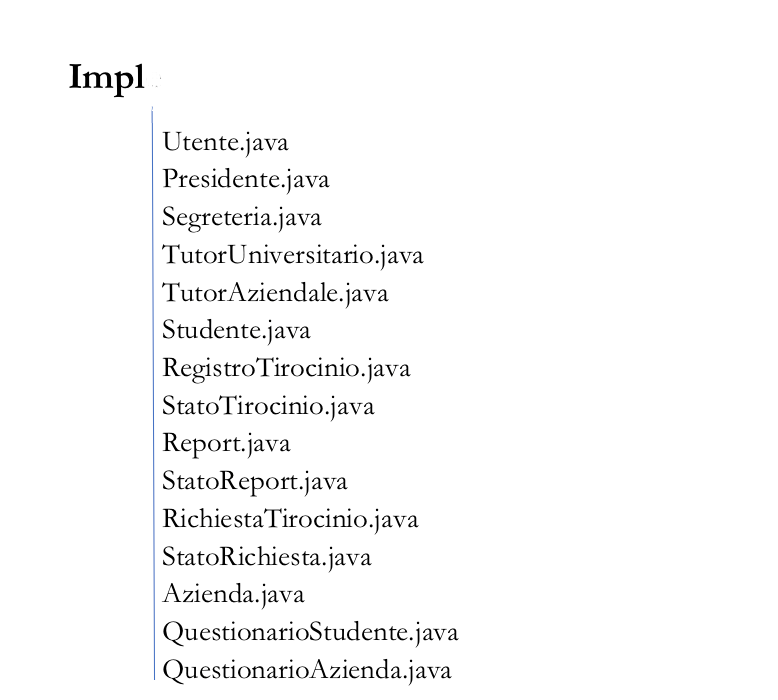
|  |  |
| --- | --- |
| Classe | Descrizione |
| UtenteBean.java | Classe che rappresenta le informazioni di un Utente. |
| PresidenteBean.java | Classe che rappresenta le informazioni del Presidente. |
| SegreteriaBean.java | Classe che rappresenta le informazioni della Segreteria. |
| TutorUniversitarioBean.java | Classe che rappresenta le informazioni del Tutor Universitario. |
| RegistroTirocinioBean.java | Classe che rappresenta le informazioni su un Registro di tirocinio. |
| StatoTirocinioBean.java | Classe che rappresenta lo stato di un tirocinio. |
| ReportBean.java | Classe che rappresenta le informazioni di un Report. |
| StatoReportBean.java | Classe che rappresenta lo stato di un Report. |
| RichiestaTirocinioBean.java | Classe che rappresenta le informazioni di una richiesta di tirocinio. |
| StatoRichiestaBean.java | Classe che rappresenta lo stato di una richiesta di tirocinio. |
| AziendaBean.java | Classe che rappresenta le informazioni di un’Azienda. |
| QuestionarioStudenteBean.java | Classe che rappresenta le informazioni di un questionario valutativo Studente. |
| QuestionarioAziendaBean.java | Classe che rappresenta le informazioni di un questionario valutazione Ente convenzionato. |

3.6 Packages DAO



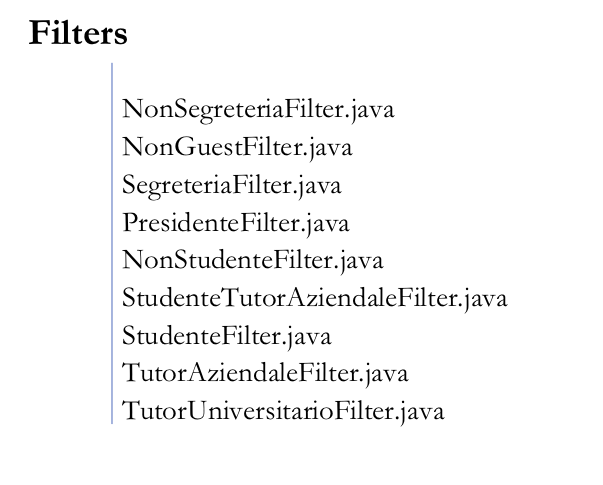
|  |  |
| --- | --- |
| Classe | Descrizione |
| GenericDAO.java | Interfaccia che definisce le operazioni CRUD comuni a tutte le entità. |
| UtenteDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali dell’Utente. |
| PresidenteDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali del Presidente |
| SegreteriaDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali della Segreteria. |
| TutorUniversitarioDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali del Tutor Universitario. |
| RegistroTirocinioDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali di un Registro di tirocinio. |
| StatoTirocinioDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali dello Stato di un tirocinio. |
| ReportDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali di un Report. |
| StatoReportDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali dello Stato di un Report. |
| RichiestaTirocinioDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali di una richiesta di Tirocinio. |
| StatoRichiestaDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali dello stato di una richiesta di tirocinio. |
| AziendaDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali per un’azienda. |
| QuestionarioStudenteDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali del questionario valutazione Studente. |
| QuestionarioAziendaDAO.java | Interfaccia che estende GenericDAO e definisce ulteriori operazioni individuali del questionario valutazione Ente Ospitante. |

3.7 Packages Impl



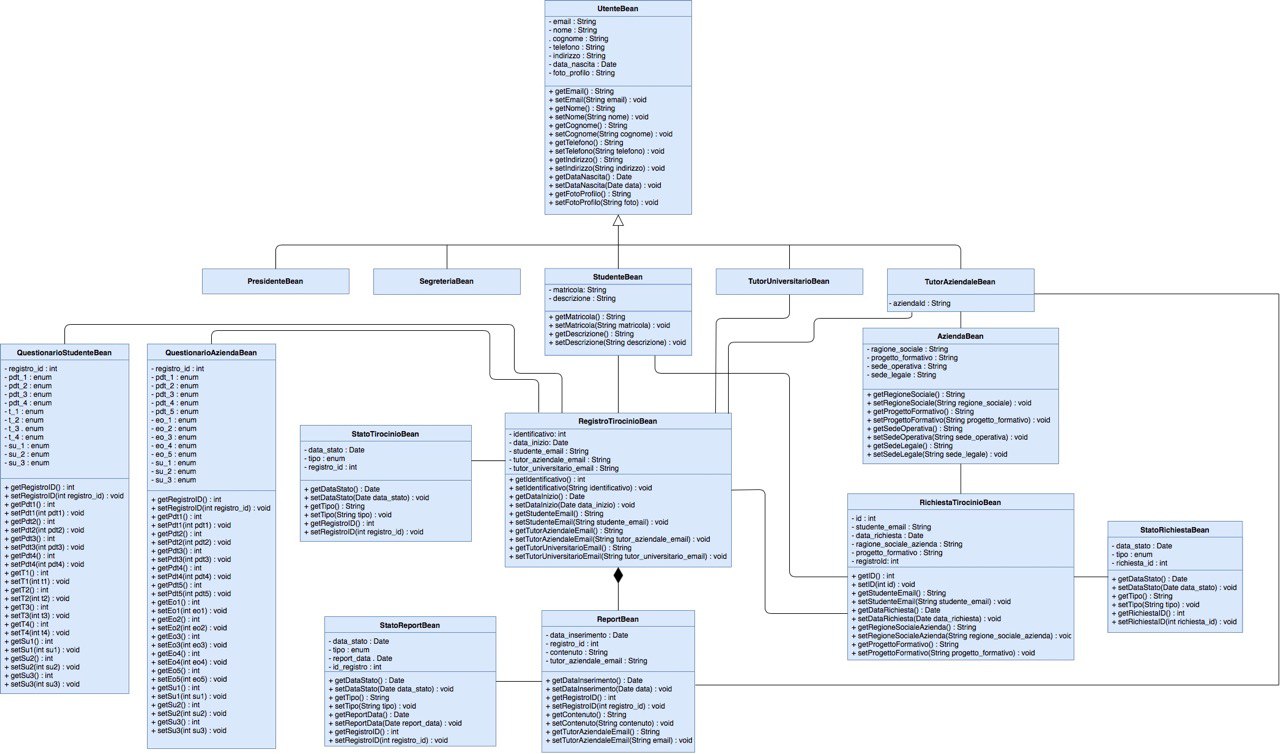
|  |  |
| --- | --- |
| Classe | Descrizione |
| Utente.java | Implementazione dell’interfaccia UtenteDAO. |
| Presidente.java | Implementazione dell’interfaccia PresidenteDAO. |
| Segreteria.java | Implementazione dell’interfaccia SegreteriaDAO. |
| TutorUniversitario.java | Implementazione dell’interfaccia TutorUniversitarioDAO. |
| RegistroTirocinio.java | Implementazione dell’interfaccia RegistroTirocinioDAO. |
| StatoTirocinio.java | Implementazione dell’interfaccia StatoTirocinioDAO. |
| Report.java | Implementazione dell’interfaccia ReportDAO. |
| StatoReport.java | Implementazione dell’interfaccia StatoReportDAO. |
| RichiestaTirocinio.java | Implementazione dell’interfaccia RichiestaTirocinioDAO. |
| StatoRichiesta.java | Implementazione dell’interfaccia StatoRichiestaDAO. |
| Azienda.java | Implementazione dell’interfaccia AziendaDAO. |
| QuestionarioStudente.java | Implementazione dell’interfaccia QuestionarioStudenteDAO. |
| QuestionarioAzienda.java | Implementazione dell’interfaccia QuestionarioAziendaDAO. |

3.8 Packages Filters



|  |  |
| --- | --- |
| Classe | Descrizione |
| NonGuestFilter.java | Filtro che concede l’accesso alla risorsa solo se l’Utente è loggato. |
| NonSegreteriaFilter.java | Filtro che nega l’accesso alla risorsa se l’Utente è Segreteria. |
| SegreteriaFilter.java | Filtro che concede l’accesso alla risorsa se l’Utente è Segreteria. |
| PresidenteFilter.java | Filtro che concede l’accesso alla risorsa se l’Utente è Presidente. |
| StudenteFilter.java | Filtro che concede l’accesso alla risorsa solo se l’Utente è Studente. |
| NonStudenteFilter.java | Filtro che concede l’accesso alla risorsa se l’Utente non è Studente. |
| TutorAziendaleFilter.java | Filtro che concede l’accesso alla risorsa solo se l’Utente è Tutor Aziendale. |
| StudenteTutorAziendaleFIlter.java | Filtro che concede l’accesso alla risorsa solo se l’Utente è Studente o Tutor Aziendale. |

4. Class Interfaces



**BEAN**

|  |  |
| --- | --- |
| Nome classe | StatoReportBean |
| Descrizione | Rappresenta le informazioni relative allo stato di un report. |
| Invarianti | - |
| Metodi | **context** StatoReportBean::getDataStato(): Date;  **context** StatoReportBean::setDataStato(dataStato: Date): void  **pre:** dataStato != null;  **context** StatoReportBean::getTipo(): String;  **context** StatoReportBean::setTipo(tipo: String): void  **pre:** tipo != null;  **context** StatoReportBean::getRegistroTirocinio(): int;  **context** StatoReportBean::setRegistroTirocinio(registroTirocinio: int): void  **pre:** registroTirocinio != null; |

|  |  |
| --- | --- |
| Nome classe | RichiestaTirocinioBean |
| Descrizione | Rappresenta le informazioni relative ad una richiesta di tirocinio. |
| Invarianti | - |
| Metodi | **context** RichiestaTirocinioBean::getID(): int;  **context** RichiestaTirocinioBean::setID(id: int): void  **pre:** id != null;  **context** RichiestaTirocinioBean::getStudente(): String;  **context** RichiestaTirocinioBean::setStudente(studente: String): void  **pre:** studenteEmail != null;  **context** RichiestaTirocinioBean::getDataRichiesta(): Date;  **context** RichiestaTirocinioBean::setDataRichiesta(dataRichiesta: Date): void  **pre:** dataRichiesta != null;  **context** RichiestaTirocinioBean::getAzienda(): String;  **context** RichiestaTirocinioBean::setAzienda(azienda: String): void  **pre:** ragioneSocialeAzienda != null;  **context** RichiestaTirocinioBean::getProgettoFormativo(): String;  **context** RichiestaTirocinioBean::setProgettoFormativo(progettoFormativo: String): void  **pre:** progettoFormativo != null;  **context** RichiestaTirocinioBean::getRegistroTirocinio(): int;  **context** RichiestaTirocinioBean::setRegistroTirocinio(registroTirocinio: int): void  **pre:** registroTirocinio != null; |

|  |  |
| --- | --- |
| Nome classe | StatoRichiestaBean |
| Descrizione | Rappresenta le informazioni relative allo stato di una richiesta di tirocinio. |
| Invarianti | - |
| Metodi | **context** StatoRichiestaBean::getDataStato(): Date;  **context** StatoRichiestaBean::setDataStato(dataStato: date): void  **pre:** dataStato != null;  **context** StatoRichiestaBean::getTipo(): String;  **context** StatoRichiestaBean::setTipo(tipo: String): void  **pre:** tipo != null;  **context** StatoRichiestaBean::getRichiestaTirocinio(): int;  **context** StatoRichiestaBean::setRichiestaTirocinio(richiestaTirocinio: int): void  **pre:** richiestaTirocinio != null; |

|  |  |
| --- | --- |
| Nome classe | AziendaBean |
| Descrizione | Rappresenta le informazioni relative a un’azienda convenzionata. |
| Invarianti | - |
| Metodi | **context** AziendaBean::getRagioneSociale(): String;  **context** AziendaBean::setRagioneSociale(ragioneSociale: String): void  **pre:** ragioneSociale != null;  **context** AziendaBean::getProgettoFormativo(): String;  **context** AziendaBean::setProgettoFormativo(progettoFormativo: String): void  **pre:** progettoFormativo != null;  **context** AziendaBean::getSedeOperativa(): String;  **context** AziendaBean::setSedeOperativa(sedeOperativa: String): void  **pre:** sedeOperativa != null;  **context** AziendaBean::getSedeLegale(): String;  **context** AziendaBean::setSedeLegale(sedeLegale: String): void  **pre:** sedeLegale != null;  **context** AziendaBean::getTutorAziendali(): String;  **context** AziendaBean::setTutorAziendali(tutorAziendali: String): void  **pre:** tutorAziendali != null;  **context** AziendaBean::getRichiesteTirocinio(): String;  **context** AziendaBean::setRichiesteTirocinio(richiesteTirocinio: String): void  **pre:** richiesteTirocinio != null; |

|  |  |
| --- | --- |
| Nome classe | QuestionarioStudenteBean |
| Descrizione | Rappresenta le informazioni relative a un questionario compilato da un tutor aziendale iscritto alla piattaforma. |
| Invarianti | - |
| Metodi | **context** QuestionarioStudenteBean::getRegistroTirocinio(): int;  **context** QuestionarioStudenteBean::setRegistroTirocinio(registroTirocinio: int): void  **pre:** registroTirocinio!=null;  **context** QuestionarioStudenteBean::getPdt1(): int;  **context** QuestionarioStudenteBean::setPdt1(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioStudenteBean::getPdt2(): int;  **context** QuestionarioStudenteBean::setPdt2(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioStudenteBean::getPdt3(): int;  **context** QuestionarioStudenteBean::setPdt3(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioStudenteBean::getPdt4(): int;  **context** QuestionarioStudenteBean::setPdt4(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioStudenteBean::getT1(): int;  **context** QuestionarioStudenteBean::setT1(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioStudenteBean::getT2(): int;  **context** QuestionarioStudenteBean::setT2(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioStudenteBean::getT3(): int;  **context** QuestionarioStudenteBean::setT3(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioStudenteBean::getT4(): int;  **context** QuestionarioStudenteBean::setT4(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioStudenteBean::getSu1(): int;  **context** QuestionarioStudenteBean::setSu1(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioStudenteBean::getSu2(): int;  **context** QuestionarioStudenteBean::setSu2(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioStudenteBean::getSu3(): int;  **context** QuestionarioStudenteBean::setSu3(i: int): void  **pre:** i > 0 && i < 6; |

|  |  |
| --- | --- |
| Nome classe | UtenteBean |
| Descrizione | Rappresenta le informazioni relative ad un utente iscritto alla piattaforma. |
| Invarianti | - |
| Metodi | **context** UtenteBean::getEmail(): String;  **context** UtenteBean::setEmail(email: String): void  **pre:** email!=null;  **context** UtenteBean::getNome(): String;  **context** UtenteBean::setNome(nome: String): void  **pre:** nome!=null;  **context** UtenteBean::getCognome(): String;  **context** UtenteBean::setCognome(cognome: String): void  **pre:** cognome!=null;  **context** UtenteBean::getTelefono(): String;  **context** UtenteBean::setTelefono(t: String): void  **pre:** t!=null;  **context** UtenteBean::getIndirizzo(): String;  **context** UtenteBean::setIndirizzo(i: String): void  **pre:** i!=null;  **context** UtenteBean::getDataNascita(): Date;  **context** UtenteBean::setDataNascita(d: Date): void  **pre:** d!=null;  **context** UtenteBean::getFotoProfilo(): String;  **context** UtenteBean::setFotoProfilo(f: String): void  **pre:** f!=null; |

|  |  |
| --- | --- |
| Nome classe | StudenteBean |
| Descrizione | Rappresenta le informazioni relative ad uno studente iscritto alla piattaforma. |
| Invarianti | - |
| Metodi | **context** StudenteBean::getMatricola(): String;  **context** StudenteBean::setMatricola(m: String): void  **pre:** m!=null;  **context** StudenteBean::getDescrizione(): String;  **context** StudenteBean::setDescrizione(d: String): void  **pre:** d!=null; |

|  |  |
| --- | --- |
| Nome classe | TutorAziendaleBean |
| Descrizione | Rappresenta le informazioni relative ad un tutor aziendale. |
| Invarianti | - |
| Metodi | **context** TutorAziendaleBean::getAziendaId(): String;  **context** TutorAziendaleBean::setAziendaId(a: String): void  **pre:** a!=null; |

|  |  |
| --- | --- |
| Nome classe | TutorUniversitarioBean |
| Descrizione | Rappresenta le informazioni relative ad un tutor Universitario. |
| Invarianti | - |
| Metodi | **context** TutorUniversitarioBean::getRegistriTirocinio(): String;  **context** TutorUniversitarioBean::setRegistriTirocinio(registriTirocinio: String): void  **pre:** registriTirocinio!=null; |

|  |  |
| --- | --- |
| Nome classe | RegistroTirocinioBean |
| Descrizione | Rappresenta le informazioni relative ad un registro di tirocinio. |
| Invarianti | - |
| Metodi | **context** RegistroTirocinioBean::getRichiestaTirocinio(): string;  **context** RegistroTirocinioBean::setRichiestaTirocinio(RichiestaTirocinio: string): void  **pre:** richiestaTirocinio!=null;  **context** RegistroTirocinioBean::getStudente(): String;  **context** RegistroTirocinioBean::setStudente(studente: String): void  **pre:** studente!=null;  **context** RegistroTirocinioBean::getTutorAziendale(): String;  **context** RegistroTirocinioBean::setTutorAziendale(tutorAziendale: String): void  **pre:** tutorAziendale!=null;  **context** RegistroTirocinioBean::getTutorUniversitario(): String;  **context** RegistroTirocinioBean::setTutorUniversitario(tutorUniversitario: String): void  **pre:** tutorUniversitario!=null; |

|  |  |
| --- | --- |
| Nome classe | StatoTirocinioBean |
| Descrizione | Rappresenta le informazioni relative ad uno stato di un tirocinio. |
| Invarianti | - |
| Metodi | **context** StatoTirocinioBean::getDataStato(): Date;  **context** StatoTirocinioBean::setDataStato(data: Date): void  **pre:** data!=null;  **context** StatoTirocinioBean::getTipo(): String;  **context** StatoTirocinioBean::setTipo(t: String): void  **pre:** t!=null;  **context** StatoTirocinioBean::getRegistroTirocinio(): int;  **context** StatoTirocinioBean::setRegistroTirocinio(registroTirocinio: int): void  **pre:** registroTirocinio!=null; |

|  |  |
| --- | --- |
| Nome classe | PresidenteBean |
| Descrizione | Rappresenta le informazioni relative al Presidente. |
| Invarianti | - |
| Metodi | - |

|  |  |
| --- | --- |
| Nome classe | ReportBean |
| Descrizione | Rappresenta le informazioni relative ad un report nel registro di tirocinio. |
| Invarianti | - |
| Metodi | **context** ReportBean::getDataInserimento(): Date;  **context** ReportBean::setDataInserimento(d: Date): void  **pre:** d!=null;  **context** ReportBean::getRegistroTirocinio(): int;  **context** ReportBean::setRegistroTirocinio(registroTirocinio: int): void  **pre:** registroTirocinio!=null;  **context** ReportBean::getPath(): String;  **context** ReportBean::setPath(path: String): void  **pre:** path!=null;  **context** ReportBean::getStatiReport(): String;  **context** ReportBean::setStatiReport(statiReport: String): void  **pre:** statiReport!=null;  **context** ReportBean::getTutorAziendaleEmail(): String;  **context** ReportBean::setTutorAziendaleEmail(email: String): void  **pre:** email!=null; |

|  |  |
| --- | --- |
| Nome classe | QuestionarioAziendaBean |
| Descrizione | Rappresenta le informazioni relative ad un questionario per un tutor aziendale. |
| Invarianti | - |
| Metodi | **context** QuestionarioAziendaBean::getRegistroId(): int;  **context** QuestionarioAziendaBean::setRegistroId(id: int): void  **pre:** id!=null;  **context** QuestionarioAziendaBean::getPdt1(): int;  **context** QuestionarioAziendaBean::setPdt1(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioAziendaBean::getPdt2(): int;  **context** QuestionarioAziendaBean::setPdt2(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioAziendaBean::getPdt3(): int;  **context** QuestionarioAziendaBean::setPdt3(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioAziendaBean::getPdt4(): int;  **context** QuestionarioAziendaBean::setPdt4(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioAziendaBean::getPdt5(): int;  **context** QuestionarioAziendaBean::setPdt5(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioAziendaBean::getEo1(): int;  **context** QuestionarioAziendaBean::setEo1(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioAziendaBean::getEo2(): int;  **context** QuestionarioAziendaBean::setEo2(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioAziendaBean::getEo3(): int;  **context** QuestionarioAziendaBean::setEo3(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioAziendaBean::getEo4(): int;  **context** QuestionarioAziendaBean::setEo4(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioAziendaBean::getEo5(): int;    **context** QuestionarioAziendaBean::setEo5(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioAziendaBean::getSu1(): int;  **context** QuestionarioAziendaBean::setSu1(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioAziendaBean::getSu2(): int;  **context** QuestionarioAziendaBean::setSu2(i: int): void  **pre:** i > 0 && i < 6;  **context** QuestionarioAziendaBean::getSu3(): int;  **context** QuestionarioAziendaBean::setSu3(i: int): void  **pre:** i > 0 && i < 6; |

**DAO**

|  |  |
| --- | --- |
| Nome classe | StatoReportDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi allo stato del report. |
| Invarianti | - |
| Metodi | **context** StatoReportDAO::getStatiReport(r: ReportBean): List<StatoReportBean>  **pre:** r != null;  **post:** result -> forAll(s: StatoReportBean | s. idRegistro == r.registroID); |

|  |  |
| --- | --- |
| Nome classe | RichiestaTirocinioDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi alla richiesta di tirocinio. |
| Invarianti | - |
| Metodi | **context** RichiestaTirocinioDAO::getRichiestaPerStato(stato: String): List<RichiestaTirocinioBean>;  **pre:** stato != null;  **post:** result -> forAll(r: RichiestaTirocinioBean | r.statoRichiestaBean.tipo.equals(stato));  **context** RichiestaTirocinioDAO::getRichiestaPerAzienda(stato: String): List<RichiestaTirocinioBean>;  **pre:** stato != null;  **post:** result -> forAll(r: RichiestaTirocinioBean | r.statoRichiestaBean.tipo.equals(stato));  **context** RichiestaTirocinioDAO::getRichiestaPerStudente(studente: StudenteBean): List<RichiestaTirocinioBean>;  **pre:** studente != null;  **post:** result -> forAll(r: RichiestaTirocinioBean | r.studenteEmail == studente.email); |

|  |  |
| --- | --- |
| Nome classe | StatoRichiestaDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi allo stato del tirocinio |
| Invarianti | - |
| Metodi | **context** StatoRichiestaDAO::getStatiRichiesta(r: RichiestaTirocinioBean): List<StatoRichiestaBean>  **pre:** r != null;  **post:** result -> forAll(s: StatoRichiestaBean | s. richiestaId == r.indentificativo); |

|  |  |
| --- | --- |
| Nome classe | AziendaDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi all’azienda convenzionata. |
| Invarianti | - |
| Metodi |  |

|  |  |
| --- | --- |
| Nome classe | PresidenteDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi al presidente. |
| Invarianti | - |
| Metodi | - |

|  |  |
| --- | --- |
| Nome classe | SegreteriaDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi alla segreteria. |
| Invarianti | - |
| Metodi | - |

|  |  |
| --- | --- |
| Nome classe | TutorUniversitarioDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi ai tutor universitari. |
| Invarianti | - |
| Metodi | - |

|  |  |
| --- | --- |
| Nome classe | QuestionarioStudenteDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi al questionario sugli studenti. |
| Invarianti | - |
| Metodi | **context** QuestionarioStudenteDAO::getQuestionarioPer(s: TutorAziendaleBean): List<QuestionarioStudenteBean>  **pre:** s != null;  **post:** result -> forAll(q: QuestionarioStudenteBean | q.registroId == s.registroTirocinio.identificativo); |

|  |  |
| --- | --- |
| Nome classe | GenericDAO |
| Descrizione | Gestisce le informazioni persistenti nel database attraverso la definizione di metodi di interrogazione, inserimento, aggiornamento e cancellazione. |
| Invarianti | - |
| Metodi | **context** GenericDAO::insert(e: Elemento): void  **pre:** e!=null;  **post:** self.retrieveAll() -> includes(e);  **context** GenericDAO::update(k: Key, e: Elemento): void  **pre:** (e!=null)&&(k!=null);  **post:** self.retrieveAll() -> includes(e);  **context** GenericDAO::retrieveByKey(k: Key): Elemento  **pre:** k!=null;  **post:** result.key == k;  **context** GenericDAO::retrieveAll(): List<Elemento>;  **context** GenericDAO::delete(k: Key): void  **pre:** k!=null;  **post:** self.retrieveByKey(k) == null; |

|  |  |
| --- | --- |
| Nome classe | UtenteDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi agli utenti. |
| Invarianti | - |
| Metodi | - |

|  |  |
| --- | --- |
| Nome classe | StudenteDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi agli studenti. |
| Invarianti | - |
| Metodi | **context** StudenteDAO::getStudentiAssociati(t: TutorAziendaleBean): List<StudenteBean>  **pre:** t!=null  **post:** result -> forAll(s: StudenteBean | s.registroTirocinio.tutorAziendaleEmail == t.email)  **context** StudenteDAO::getStudentiAssociati(t: TutorUniversitarioBean): List<StudenteBean>  **pre:** t!=null  **post:** result -> forAll(s: StudenteBean | s.registroTirocinio.tutorUniversitarioEmail == t.email) |

|  |  |
| --- | --- |
| Nome classe | TutorAziendaleDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi ai tutor aziendali. |
| Invarianti | - |
| Metodi | **context** StudenteDAO::getTutorDiAzienda(a: AziendaBean): List<TutorAziendaleBean>  **pre:** a!=null  **post:** result -> forAll(t: TutorAziendaleBean |t.aziendaId.equals(a.azienda)); |

|  |  |
| --- | --- |
| Nome classe | RegistroTirocinioDAO |
| 5Descrizione | Definisce i metodi per gestire i dati relativi ai registri di tirocinio. |
| Invarianti | - |
| Metodi | **context** RegistroTirocinioDAO::getRegistriDiStudente(s: StudenteBean): List<RegistroTirocinioBean>  **pre:** s!=null;  **post:** result ->forAll(r: RegistroTirocinioBean | r.studenteEmail == s.email)  **context** RegistroTirocinioDAO::getRegistriDiTutorAziendale(t: TutorAziendaleBean): List<RegistroTirocinioBean>  **pre:** t!=null;  **post:** result -> forAll(r: RegistroTirocinioBean | r.tutorAziendaleEmail.equals(t.email));  **context** RegistroTirocinioDAO::getRegistriDiTirociniInCorso(): List<RegistroTirocinioBean>;  **post:** result -> forAll(r: RegistroTirocinioBean | r.statoTirocinio.tipo == “in corso”)  **context** RegistroTirocinioDAO::getRegistriDiTirociniTerminati(): List<RegistroTirocinioBean>;  **post:** result -> forAll(r: RegistroTirocinioBean | r.statoTirocinio.tipo == “terminato”)  **context** RegistroTirocinioDAO::getRegistriDiTutorUniversitario(): List<RegistroTirocinioBean>;  **post:** result -> forAll(r: RegistroTirocinioBean | r.tutorUniversitarioEmail.equals(t.email)); |

|  |  |
| --- | --- |
| Nome classe | StatoTirocinioDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi agli stati di tirocinio. |
| Invarianti | - |
| Metodi | **context** StatoTirocinioDAO::getStatiDiTirocinio(r: RegistroTirocinioBean): List<StatoTirocinioBean>  **pre:** r!=null;  **post:** result -> forAll(s: StatoTirocinioBean | s.registroId == r.indentificativo); |

|  |  |
| --- | --- |
| Nome classe | ReportDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi ai report. |
| Invarianti | - |
| Metodi | **context** ReportDAO::getReportFirmati(reg: RegistroTirocinioBean): List<ReportBean>;  **pre:** reg != null;  **post:** result -> forAll(r: ReportBean | r.registroId == reg.identificativo && t.tutorAziendaleEmail == reg.tutorAziendaleEmail)  **context** ReportDAO::getReportNonFirmati(reg: RegistroTirocinioBean): List<ReportBean>  **pre:** reg != null;  **post:** result -> forAll(r: ReportBean | r.registroId == reg.identificativo && t.tutorAziendaleEmail == null);  **context** ReportDAO::getReportRegistro(reg: RegistroTirocinioBean): List<ReportBean>  **pre:** r!=null;  **post:** result -> forAll(r: ReportBean | r.registroId == reg.identificativo); |

|  |  |
| --- | --- |
| Nome classe | QuestionarioAziendaDAO |
| Descrizione | Definisce i metodi per gestire i dati relativi ai questionari per gli studenti. |
| Invarianti | - |
| Metodi | **context** QuestionarioAziendaDAO::getQuestionarioPer(s: StudenteBean): List<QuestionarioAziendaBean>  **pre:** s!=null;  **post:** result -> forAll(q: QuestionarioAziendaBean | q.registroId == s.registroTirocinio.identificativo); |

**Facade**

|  |  |
| --- | --- |
| Nome classe | GestioneAccountFacade |
| Descrizione | Definisce i metodi per gestire le funzionalità relative all’account. |
| Invarianti | - |
| Metodi | **context** GestioneAccountFacade::login(email: String, pwd: String): UtenteBean  **pre:** (email!=null)&&(pwd!=null)  **post:** (utente.getEmail() == email)  **context** GestioneAccountFacade::registrazione(s: StudenteBean, pwd: String): boolean  **pre:** (s!=null)&&(pwd!=null)  **post:** StudenteDAO.retrieveAll() -> includes(s) |

|  |  |
| --- | --- |
| Nome classe | GestioneAreaPersonaleFacade |
| Descrizione | Definisce i metodi per gestire le funzionalità relative all’area personale. |
| Invarianti | - |
| Metodi | **context** GestioneAreaPersonaleFacade::modificaProfilo(u: UtenteBean): boolean  **pre:** u!=null;  **post:** UtenteDAO.retrieveByKey(u.getEmail()).getFotoProfilo() == u.getFotoProfilo();  AND  UtenteDAO.retrieveByKey(u.getEmail()).getTelefono() == u.getTelefono();  AND  UtenteDAO.retrieveByKey(u.getEmail()).getIndirizzo() == u.getIndirizzo();    **context** GestioneAreaPersonaleFacade::modificaProfilo(s: StudenteBean): boolean  **pre:** s!=null;  **post:** StudenteDAO.retrieveByKey(s.getEmail()).getDescrizione() == s.getDescrizione();  **context** GestioneAreaPersonaleFacade::listaStudentiAssociati(t: TutorAziendaleBean): List<StudenteBean>  **pre:** t!=null;  **post:** result -> forAll(s: StudenteBean | s.registroTirocinio.tutorAziendaleEmail == t.email);  **context** GestioneAreaPersonaleFacade::listaStudentiAssociati(t: TutorUniversitarioBean): List<StudenteBean>  **pre:** t!=null;  **post:** result -> forAll(s: StudenteBean | s.registroTirocinio.tutorUniversitarioEmail == t.email); |

|  |  |
| --- | --- |
| Nome classe | GestionePraticheTirocinioFacade |
| Descrizione | Definisce i metodi per gestire le funzionalità relative alle pratiche di tirocinio. |
| Invarianti | - |
| Metodi | **context** GestionePraticheTirocinioFacade::listaAziende(): List<AziendaBean>  **context** GestionePraticheTirocinioFacade::inserisciAzienda(a: AziendaBean): boolean  **pre:** a!= null;  **post:** AziendaDAO.retrieveAll() -> includes(a);  **context** GestionePraticheTirocinioFacade::aggiornaAzienda(a: AziendaBean): boolean  **pre:** a!=null;  **post:** AziendaDao.retrieveByKey(a.getRagioneSociale()).getSedeOperativa() == a.getSedeOperativa();  AND  AziendaDao.retrieveByKey(a.getRagioneSociale()).getSedeLegale() == a.getSedeLegale();  **context** GestionePraticheTirocinioFacade::listaStudenti(): List<StudenteBean>  **context** GestionePraticheTirocinioFacade::inviaRichiestaTirocinio(r: RichiestaTirocinioBean): boolean  **pre:** r!=null AND !(RichiestaTirocinioDAO.retrieveAll() -> includes(r));  **post:** RichiestaTirocinioDAO.retrieveAll() -> includes(r)  AND  StatoTirocinioDAO.getStatiDiRichiesta(r) -> exists(s: StatoRichiestaBean | s.tipo.equals(“nuova”));  **context** GestionePraticheTirocinioFacade::listaRichiesteTirocinio(): List<RichiestaTirocinioBean>    **context** GestionePraticheTirocinioFacade::valutazioneInizialeRichiesta(r: RichiestaTirocinioBean): boolean  **pre:** r!=null AND RichiestaTirocinioDAO.retrieveAll() -> includes(r);  **post:**  StatoTirocinioDAO.getStatiRichiesta(r) -> exists(s: StatoRichiestaBean | s.tipo.equals(“in validazione”))  AND  RegistroTirocinioDAO.retrieveAll() -> includes(r.registroTirocinio);  **context** GestionePraticheTirocinioFacade::valutazioneFinaleRichiesta(r: RichiestaTirocinioBean): boolean  **pre:** r!=null AND RichiestaTirocinioDAO.retrieveAll() -> includes(r);  **post:**  StatoTirocinioDAO.getStatiRichiesta(r) -> exists(s: StatoRichiestaBean | s.tipo.equals(“in convalida”));  **context** GestionePraticheTirocinioFacade::convalidaRichiesta(r: RichiestaTirocinioBean): boolean  **pre:** r!=null AND RichiestaTirocinioDAO.retrieveAll() -> includes(r);  **post:**  StatoTirocinioDAO.getStatiRichiesta(r) -> exists(s: StatoRichiestaBean | s.tipo.equals(“accettata”));  **context** GestionePraticheTirocinioFacade::assegnaTutor(ta: TutorAziendaleBean, tu: TutorUniversitarioBean, r: RichiestaTirocinioBean): boolean  **pre:** tu!=null AND ta!=null  AND  TutorUniversitarioDAO.retrieveAll() - > includes(tu)  AND  TutorAziendaleDAO.retrieveAll() -> includes(ta)  AND  StatoRichiestaDAO.getStatiRichiesta(r) -> exists(s: StatoRichiestaBean | s.tipo.equals(“nuova”))  AND  RegistroTirocinioDAO.retrieveAll() -> includes(r.registroTirocinio);  **post:** r.registroTirocinio.tutorAziendaleEmail.equals(ta.email)  AND  r.registroTirocinio.tutorUniversitarioEmail.equals(tu.email); |

|  |  |
| --- | --- |
| Nome classe | GestioneReportFacade |
| Descrizione | Definisce i metodi per gestire le funzionalità relative ai report. |
| Invarianti | - |
| Metodi | **context** GetioneReportFacade::listaRegistroTirocinio(): List<RegistroTirocinioBean>  **context** GetioneReportFacade::listaRegistroTirocinio(tutor: TutorAziendaleBean): List<RegistroTirocinioBean>  **pre:** tutor != null  **post:** result -> forAll(r: RegistroTirocinioBean | r.tutorAziendaleEmail.equals(tutor.email));  **context** GetioneReportFacade::listaRegistroTirocinio(tutor: TutorUniversitarioBean): List<RegistroTirocinioBean>  **pre:** tutor != null  **post:** result -> forAll(r: RegistroTirocinioBean | r.tutorUniversitarioEmail.equals(tutor.email));  **context** GetioneReportFacade::inserimentoReport(report: ReportBean, registro: RegistroTirocinioBean): void  **pre:** report != null && registro != null  **post:** ReportDAO.getReportRegistro(registro) -> includes(report);  **context** GetioneReportFacade::recuperaQuestionarioStudente(tutor: TutorAziendaleBean): List<QuestionarioStudenteBean>  **pre:** tutor != null  **post:** result -> forAll(q: QuestionarioStudenteBean | RegistroTirocinioDAO.getRegistriDITutorAziendale(tutor) -> exists(r: RegistroTirocinioBean | r.identificativo == q.registroId))  **context** GetioneReportFacade::recuperaQuestionarioAzienda(studente: StudenteBean): List<QuestionarioAziendaBean>  **pre:** studente != null  **post:** result -> forAll(q: QuestionarioAziendaBean | RegistroTirocinioDAO.getRegistriDIStudente(studente) -> exists(r: RegistroTirocinioBean | r.identificativo == q.registroId))  **context** GetioneReportFacade::inserisciQuestionarioStudente(questionario: QuestionarioStudenteBean): void  **pre:** questionario != null  **post:** QuestionarioStudenteDAO.retriveAll() -> includes(questionario);  **context** GetioneReportFacade::inserisciQuestionarioAzienda(questionario: QuestionarioAziendaBean): void  **pre:** questionario != null  **post:** QuestionarioAziendaDAO.retriveAll() -> includes(questionario); |

|  |  |
| --- | --- |
| Nome classe | GestioneModulisticaFacade |
| Descrizione | Definisce i metodi per gestire le funzionalità relative alla modulistica. |
| Invarianti | - |
| Metodi | **context** GestioneModulisticaFacade::downloadProgettoFormativo(azienda: AziendaBean): String  **pre:** aziendaBean != null  **post**: aziendaBean.getProgettoFormativo().equals(result);  **context** GestioneModulisticaFacade::downloadRichiestaTirocinio(richiesta: RichiestaTirocinioBean): String  **pre:** richiesta != null  **post**: richiesta.getProgettoFormativo().equals(result);  **context** GestioneModulisticaFacade::downloadReport(report: ReportBean): String  **pre:** report != null  **post**:report.getContenuto().equals(result);  **context** GestioneModulisticaFacade::uploadProgettoFormativo(file: File, azienda: AziendaBean): void  **pre:** file != null && azienda != null  **post**: AziendaDAO.retrieveByKey(azienda.getRagioneSociale()).getProgettoFormativo() == file.path  **context** GestioneModulisticaFacade::uploadRichiestaTirocinio(file: File, richiesta: RichiestaTirocinioBean): void  **pre:** file != null && richiesta != null  **post**: RichiestaDAO.retrieveByKey(richiesta.getId()).getProgettoFormativo() == file.path  **context** GestioneModulisticaFacade::uploadReport(file: File, richiesta: ReportBean): void  **pre:** file != null && richiesta != null  **post**: ReportDAO. retrieveByKey(report.getRegistroId(), report.getDataInserimento).getContenuto() == file.path |

**Servlet**

|  |  |
| --- | --- |
| Nome classe | GeneraleServlet |
| Descrizione | Gestisce le richieste generate dall’utente interagendo con l view. |
| Invarianti | - |
| Metodi | **context** GeneraleServlet::doGet(request: HttpServletRequest, response: HttpServletResponse)  **pre:** request!=null AND response!=null;  **context** GeneraleServlet::doPost(request: HttpServletRequest, response: HttpServletResponse)  **pre:** request!=null AND response!=null; |