



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



Object Design Document

Easy Traineeship

Riferimento	
Versione	1.0
Data	12/01/2020
Destinatario	Top Management
Presentato da	Alessandro Serritiello, Elio Testa, Felice Montella, Giuseppe Barisano, Luigi Barbato, Pellegrino Aurilia, Salvatore Amendola, Simona Grilletto
Approvato da	Vincenzo Belmonte, Vincenzo De Rosa



Revision History

Data	Versione	Descrizione	Autori
06/12/2019	0.1	Inizio stesura ODD	Giuseppe Barisano
06/12/2019	0.1	Aggiunta dei capitoli 1, 1.1	Salvatore Amendola/ Giuseppe Barisano/ Felice Montella/ Alessandro Serritiello
06/12/2019	0.1	Aggiunta del capitolo 1.2	Pellegrino Aurilia/ Luigi Barbato/ Simona Grilletto/ Elio Testa
06/12/2019	0.1	Aggiunta del Capitolo 4.1	[Tutti]
09/12/2019	0.2	Aggiunta del capitolo 2	Barisano Giuseppe/ Barbato Luigi/ Felice Montella/ Simona Grilletto/ Alessandro Serritiello
12/12/2019	0.3	Aggiunta del Capitolo 3	[Tutti]
12/12/2019	0.3	Aggiunta del Capitolo 4.2	Pellegrino Aurilia/ Salvatore Amendola/ Elio Testa
13/12/2019	0.4	Stesura documento	Giuseppe Barisano
13/12/2019	0.4	Revisione documento	Simona Grilletto
10/10/2020	0.5	Revisione del capitolo: 1.1	Alessandro Serritiello/ Salvatore Amendola
11/01/2020	1.0	Revisione dei Capitoli: 2, 4.1.1, 4.2	Felice Montella/ Pellegrino Aurilia/ Giuseppe Barisano



Sommario

Revision History	2
1. Introduzione.....	5
1.1. Object design Trade-off	5
1.2. Interface Documentation Guidelines	6
1.2.1. Componenti off-the-shelf	6
1.2.2. Classi e Interfacce Java.....	6
1.2.3. Pagine lato Server (JSP).....	7
1.2.4. Pagine HTML.....	7
1.2.5. Script Javascript	7
1.2.6. Fogli di stile CSS	8
1.2.7. Database SQL.....	8
1.3. Definizione, acronimi e abbreviazioni	9
1.4. Riferimenti.....	9
1.5. Organizzazione del documento.....	9
2. Packages.....	9
2.1. Package Model	10
2.2. Package Control	11
2.3. Package <i>WebContent</i>	14
3. Class interfaces.....	17
4. Design Pattern con Class Diagram	28
4.1. Design Pattern	28
4.1.1. Design Pattern DAO (Data Access Objects)	28
4.1.2. Design Pattern Singleton	29
4.2. Class Diagram	30



5. Glossario.....	30
-------------------	----



1. Introduzione

Dopo aver stilato il documento di “*Requirements Analysis*” e il documento di “*System Design*”, è necessario concentrare l’attenzione sugli aspetti implementativi. Il documento di “*Object Design*”, che verrà redatto, ha come obiettivo quello di produrre un modello che sia in grado di integrare in un unico insieme, in modo coerente e preciso, tutte le informazioni collezionate durante le fasi precedenti.

In particolar modo, in tale documento verranno definite le interfacce delle classi, le operazioni supportate, i tipi dei dati, i parametri delle procedure, i signatures dei sottosistemi definiti nel documento di System Design, i trade-off e le linee guida.

1.1. Object design Trade-off

- ***Attendibilità vs Tempi di risposta***

Il Sistema sarà implementato in modo tale da preferire l’attendibilità al tempo di risposta, in modo tale da garantire un controllo più accurato dei dati in input a discapito del tempo di risposta del sistema.

- ***Disponibilità vs Tolleranza ai guasti***

Il Sistema deve sempre essere disponibile all’utente in caso di errore in una funzionalità, anche al costo di rendere non disponibile quest’ultima per un lasso di tempo.

- ***Memoria vs Estendibilità***

Il sistema deve permettere l’estendibilità a discapito della memoria utilizzata. Tale preferenza permette al cliente di richiedere agli sviluppatori nuove funzionalità, dando meno importanza alla memoria utilizzata.

- ***Criteri di Manutenibilità vs Criteri di Performance***

Il Sistema sarà implementato preferendo la manutenibilità ai tempi di risposta in modo da facilitare gli sviluppatori nel processo di aggiornamento del software a discapito delle performance del sistema

- ***Comprensibilità vs Tempo***

Il codice del sistema deve essere comprensibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare. Al fine di rispettare queste linee guida il codice sarà integrato da commenti volti a migliorarne la leggibilità; tuttavia questo richiederà una maggiore quantità di tempo necessario per lo sviluppo del nostro progetto.



1.2. Interface Documentation Guidelines

1.2.1. Componenti off-the-shelf

Il Sistema farà uso di componenti off-the-shelf, per cui si useranno i framework adibiti al riuso di componenti già esistenti adibiti alla facilitazione del lavoro implementativo.

Bootstrap: Framework d'utilizzo per la parte front-end che comprende: HTML&CSS.

JQuery: Framework d'utilizzo per la parte front-end che comprende: JAVASCRIPT & AJAX.

1.2.2. Classi e Interfacce Java

Nella scrittura di codice per le classi Java ci si atterrà allo standard Google Java (<http://google.github.io/styleguide/javaguide.html#s4.6-whitespace>) nella sua interezza. Tale standard fornisce delle regole da seguire ad esempio ogni metodo ed ogni file possono non essere preceduti da un commento. Potranno esserci, inoltre, commenti e giustificazioni in merito a particolari decisioni o calcoli. La convenzione utilizzata dai team member per quanto riguarda i nomi delle variabili, è la nota lowerCamelCase, che consiste nello scrivere parole composte o frasi unendo tutte le parole tra loro. Quando si codificano classi e interfacce Java, si dovrebbero rispettare le seguenti regole di formattazione:

- Non inserire spazi tra il nome del metodo e la parentesi tonda “(” che apre la lista dei parametri.
- La parentesi graffa aperta “{” si trova alla fine della stessa linea dell’istruzione di dichiarazione.
- La parentesi graffa chiusa “}” inizia su una nuova riga vuota allo stesso livello di indentazione del nome della classe o dell’interfaccia.

Nel caso di istruzioni semplici, ogni linea deve contenere al massimo una sola istruzione. Mentre nel caso di istruzioni composte vanno rispettate le seguenti regole:

- Le istruzioni racchiuse all’interno di un blocco(esempio: for), devono essere indentate di un’unità all’interno dell’istruzione composta.
- La parentesi di apertura del blocco deve trovarsi alla fine della riga dell’istruzione composta.
- La parentesi di chiusura del blocco deve trovarsi allo stesso livello di indentazione dell’istruzione composta
- Le istruzioni composte formate da un’unica istruzione devono essere racchiuse da parentesi.



I nomi di classe devono essere sostantivi, con lettere minuscole e, sia la prima lettera del nome della classe sia la prima lettera di ogni parola interna, deve essere maiuscola. I nomi delle classi dovrebbero essere semplici, descrittivi e che rispettino il dominio applicativo. Non dovrebbero essere usati underscore per legare nomi. I nomi dei metodi iniziano con una lettera minuscola (non sono consentiti caratteri speciali) e seguono la notazione a cammello. Dovranno essere semplici, descrittivi e che rispettino il dominio applicativo.

1.2.3. Pagine lato Server (JSP)

Le pagine JSP devono, quando eseguite, produrre un documento conforme allo standard HTML 5. Il codice Java delle pagine deve aderire alle convenzioni per la codifica in Java, con le seguenti puntualizzazioni:

- Il tag di apertura (<%) è seguito immediatamente dalla fine della riga;
- Il tag di chiusura (%>) si trova all'inizio di una riga;

Per le Servlet è necessario far terminare il nome della classe con il suffisso Servlet.

1.2.4. Pagine HTML

Le pagine HTML, sia in forma statica che dinamica, devono essere conformi allo standard HTML 5. Inoltre, il codice HTML statico deve utilizzare l'indentazione, per facilitare la lettura, secondo le seguenti regole:

- Un'indentazione consiste in una tabulazione;
- Ogni tag deve avere un'indentazione maggiore del tag che lo contiene;
- Ogni tag di chiusura deve avere lo stesso livello di indentazione del corrispondente tag di apertura;
- I tag di commento devono seguire le stesse regole che si applicano ai tag normali.

1.2.5. Script Javascript

Gli Script in Javascript devono rispettare le seguenti convenzioni:

Gli Script che svolgono funzioni distinte dal mero rendering della pagina dovrebbero essere collocati in file dedicati.

- Il codice Javascript deve seguire le stesse convenzioni per il layout e i nomi del codice Java.



- I documenti Javascript devono essere iniziati da un commento analogo a quello presente nei file Java.
- Le funzioni Javascript devono essere documentate in modo analogo ai metodi Java.
- Gli oggetti Javascript devono essere preceduti da un commento in stile Javadoc, che segue il seguente formato:

1.2.6. Fogli di stile CSS

- Non è ammesso CSS inline
- Ogni foglio di stile deve essere iniziato da un commento analogo a quello presente nei file Java.
- Ogni regola CSS deve essere formattata come segue: I selettori della regola si trovano a livello 0 di indentazione, uno per riga;
- L'ultimo selettore della regola è seguito da parentesi graffa aperta ({});
- Le proprietà che costituiscono la regola sono listate una per riga e sono indentate rispetto ai selettori; La regola è terminata da una parentesi graffa chiusa (}), collocata da sola su una riga;

1.2.7. Database SQL

I nomi delle tabelle devono seguire le seguenti regole:

- Devono essere costituiti da sole lettere maiuscole;
- Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.

I nomi dei campi devono seguire le seguenti regole:

- Devono essere costituiti da sole lettere maiuscole;
- Se il nome è costituito da più parole, è previsto l'uso di underscore (_);
- Il nome deve essere un sostantivo singolare tratto dal dominio del problema ed esplicativo del contenuto.



1.3. Definizione, acronimi e abbreviazioni

1.4. Riferimenti

- Bernd Bruegge & Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, (2nd edition), Prentice-Hall, 2003
- Ian Sommerville, Software Engineering, Addison Wesley
- Statement of work
- English Validation, Sistema di validazione di UNISA
- C2_RAD_Vers. 1.3.pdf
- C2_SDD_Vers. 0.7.pdf
- C2_ClassDiagram_ODD_Vers 0.1.jpg

1.5. Organizzazione del documento

Introduzione: Contiene una panoramica di quello che verrà stilato nel documento di “*Object Design*”, i suoi obiettivi, le informazioni riguardanti l’implementazione, gli “*Object design Trade-off*”, “*Interface Documentation Guidelines*” e un elenco di definizioni, acronimi e abbreviazioni utili alla comprensione dell’intera documentazione.

Packages: Contiene la suddivisione del Sistema nei vari “*Packages*” riscontrati, con relative informazioni.

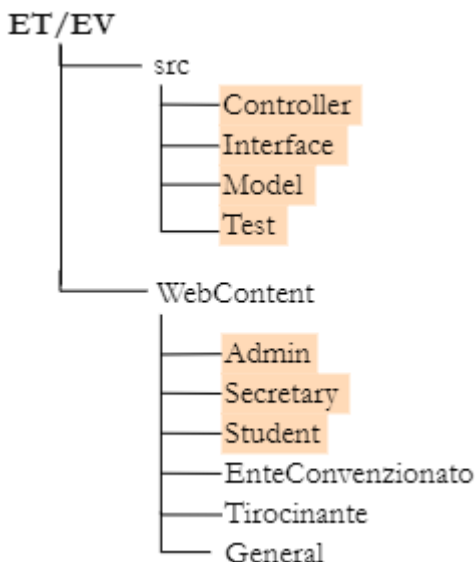
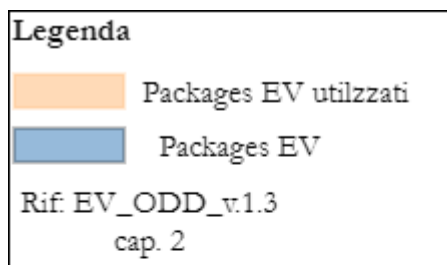
Class interfaces: Contiene la descrizione delle classi del Sistema nel dettaglio con pre e post condizione.

Design Pattern con Class Diagram: Contiene i “*Design Pattern*” scelti per il nostro Sistema, con relativa descrizione e motivazione. Inoltre, contiene il “*Class Diagram*” il quale contiene la panoramica delle classi del nostro Sistema.

Glossario: Raccolta di vocaboli meno comuni utilizzati nella stesura del documento.

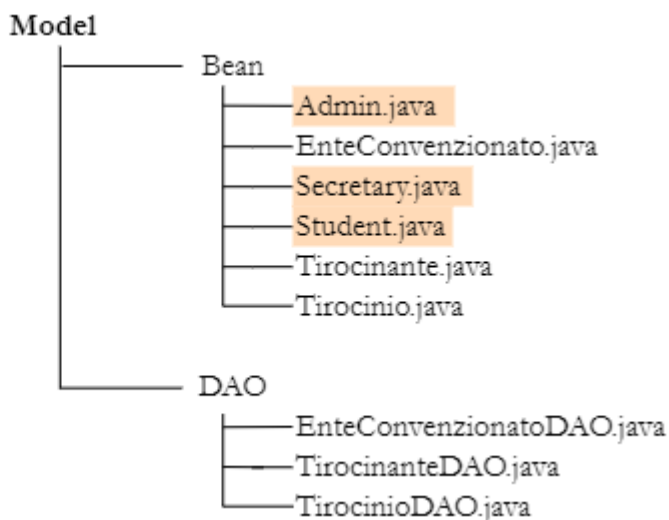
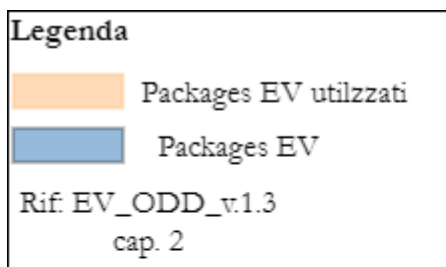
2. Packages

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. Il Sistema corrente non prevede l’utilizzo dei “*DAO*” (utilizzati per l’accesso ai dati, mediante le query, in base alle operazioni da effettuare), mentre, il Sistema Proposto andrà ad integrare questi ultimi, con l’utilizzo dei “*Bean*” (utilizzati per incapsulare più oggetti in un singolo bin, cosicché è possibile utilizzare un singolo oggetto) precedentemente utilizzati.



Classe	Descrizione
Admin.java	Classe che rappresenta le informazioni dell'Admin.



EnteConvenzionato.java	Classe che rappresenta le informazioni dell'Ente Convenzionato.
EnteConvenzionatoDAO.java	Classe che accede alle informazioni dell'Ente Convenzionato.
Secretary.java	Classe che rappresenta le informazioni della Segreteria.
Student.java	Classe che rappresenta le informazioni dello Studente.
Tirocinante.java	Classe che rappresenta le informazioni del Tirocinante.
TirocinanteDAO.java	Classe che accede alle informazioni del Tirocinante.
Tirocinio.java	Classe che rappresenta le informazioni del Tirocinio.
TirocinioDAO.java	Classe che accede alle informazioni del Tirocinio.

2.2. Package Control

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.

All’interno di questo Package, sono riportate le Servlet già esistenti, alcune delle quali verranno riusate per funzionalità comuni al Sistema Proposto; altre, invece, verranno implementate per ricoprire le nuove funzionalità del Sistema.



Legenda	
	Packages EV utilizzati
	Packages EV
Rif: EV_ODD_v1.3 cap. 2	

Controller

—	CheckSession.java
—	DbConnection.java
—	Downloader.java
—	SendMail
—	ServletAdmin.java
—	ServletAnnullaEnteDaStudenteET.java
—	ServletAnnullaTirocinioET.java
—	ServletCommon.java
—	ServletDocumentiTirocinioET.java
—	ServletDownloadET.java
—	ServletEliminaEnteET.java
—	ServletGestioneRichiesteEnteET.java
—	ServletGestioneRichiesteSegreteriaET.java
—	ServletListaEnteET.java
—	ServletModificaEnteET.java
—	ServletProgettoFormativoET.java
—	ServletRegistrazioneEnteET.java
—	ServletRichiestaInizioTirocinioET.java
—	ServletSceltaEnteET.java
—	ServletSecretary.java
—	ServletStatoTirocinanteET.java
—	ServletStatoTirocinioET.java
—	ServletStudent.java
—	ServletUploadET.java
—	ServletVisualizzaListaTirocinantiET.java
—	ServletVisualizzaTirocinanteEnteET.java
—	ServletVisualizzaTirocinanteET.java
—	Uploader.java
—	Utils.java

Classe	Descrizione
CheckSession.java	Classe per il controllo della sessione.
DbConnection.java	Classe che crea la connessione con il DB.
Downloader.java	Classe per la gestione dei file da scaricare per "English Validation"
ServletAdmin.java	Classe per la gestione delle richieste dell'Admin per "English Validation"
ServletAnnullaEnteDaStudenteET.java	Classe per la gestione dell'annullamento della richiesta all'Ente da parte dello Studente



ServletAnnullaTirocinioET.java	Classe per la gestione dell'annullamento del Tirocinio
ServletDocumentiTirocinioET.java	Classe per la gestione dei documenti
ServletDownloadET.java	Classe per la gestione del download del Progetto Formativo.
ServletGestioneRichiesteEnteET.java	Classe per la gestione delle richieste per l'Ente
ServletGestioneRichiesteSegreteriaET.java	Classe per la gestione delle richieste per la Segreteria
ServletRichiestaInizioTirocinioET.java	Classe per la gestione della richiesta di inizio Tirocinio da parte dello Studente
ServletSceltaEnteET.java	Classe per la gestione della richiesta dell'Ente da parte dello Studente
ServletStatoTirocinanteET.java	Classe per la gestione dello Stato del Tirocinio da parte dello Studente
ServletStatoTirocinioET.java	Classe per la gestione dello Stato del Tirocinio da parte della Segreteria
ServletVisualizzaListaTirocinantiET.java	Classe per la gestione della lista dei Tirocinanti
ServletCommon.java	Classe per la gestione del login/modifica.
ServletListaEnteET.java	Classe per la gestione della lista degli Enti
ServletRegistrazioneEnteET.java	Classe per la gestione della registrazione dell'Ente da parte della Segreteria
ServletModificaEnteET.java	Classe per la gestione della modifica dell'Ente da parte della Segreteria
ServletEliminaEnteET.java	Classe per la gestione dell'eliminazione dell'Ente da parte della Segreteria
ServletProgettoFormativoET.java	Classe per la gestione del Progetto Formativo.
ServletRecuperoPasswordET.java	Classe per la gestione del recupero password
ServletStudent.java	Classe per la gestione della registrazione per lo Studente.
ServletVisualizzaTirocinanteET.java	Classe per la gestione del Tirocinante
ServletVisualizzaTirocinanteEnteET.java	Classe per la gestione del Tirocinante da parte dell'Ente
ServletSecretary.java	Classe per la gestione delle richieste della Segreteria per "English Validation"
ServletUploadET.java	Classe per la gestione del download del Progetto Formativo.



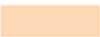
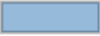
Uploader.java	Classe per la gestione dei file da caricare per "English Validation"
Utils.java	Classe per la gestione della cifratura delle password

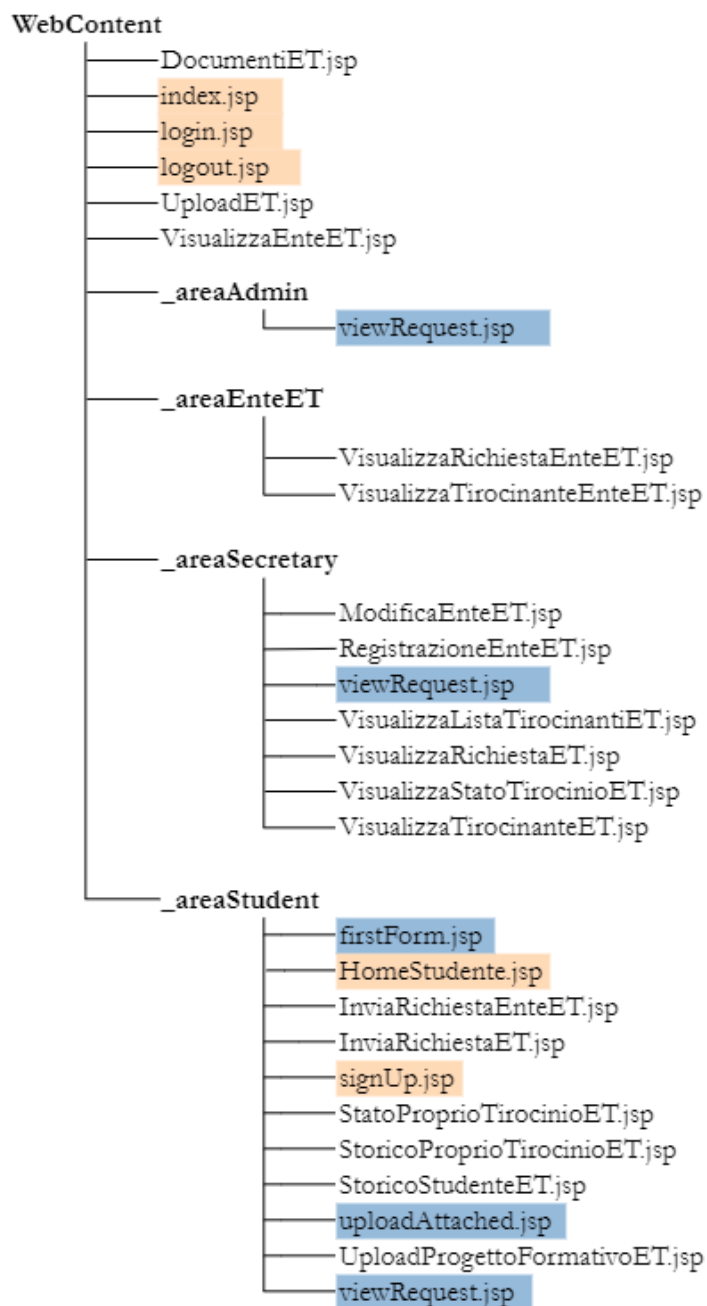
2.3. Package *WebContent*

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.

Alcune delle pagine JSP già esistenti nel Sistema di "*English Validation*" verranno riutilizzate per funzioni comuni all'applicazione Corrente, mentre altre che dovranno contenere le nuove funzionalità, verranno implementare.



Legenda	
	Packages EV utilizzati
	Packages EV
Rif: EV_ODD_v.1.3 cap. 2	



Classe	Descrizione
DocumentiET.jsp	Pagina per la visualizzazione dei documenti da firmare.
index.jsp	Pagina principale per l'utente.
HomeStudiante.jsp	Pagina dell'area utente per lo Studente.
viewRequest.jsp	Pagina delle richieste per l'utente.



InviaRichiestaEnteET.jsp	Pagina per la richiesta di inizio Tirocinio dello Studente verso l'Ente Convenzionato.
InviaRichiestaET.jsp	Pagina per la richiesta di Inizio Tirocinio dello Studente.
login.jsp	Pagina per la gestione delle operazioni di login.
logout.jsp	Pagina per la gestione delle operazioni di logout.
RecuperaPassword.jsp	Pagina per il recupero della password
ModificaEnteET.jsp	Pagina per la modifica dell'Ente Convenzionato da parte della Segreteria.
RegistrazioneEnteET.jsp	Pagina per la registrazione dell'Ente Convenzionato da parte della Segreteria.
signUp.jsp	Pagina per la registrazione da parte dello Studente.
VisualizzaEnteET.jsp	Pagina per la visualizzazione della lista degli Enti.
VisualizzaTirocinanteET.jsp	Pagina per visualizzazione della lista dei Tirocinanti.
VisualizzaListaTirocinantiET.jsp	Pagina per la visualizzazione della lista dei Tirocinanti.
VisualizzaRichiestaET.jsp	Pagina per la visualizzazione delle richieste di inizio Tirocinio.
VisualizzaStatoTirocinioET.jsp	Pagina per la visualizzazione dello stato di Tirocinio
StatoProprioTirocinioET.jsp	Pagina per la visualizzazione dello stato di Tirocinio di uno Studente.
VisualizzaRichiestaEnteET.jsp	Pagina per la visualizzazione delle richieste da parte dell'Ente.
UploadET.jsp	Pagina per caricare il progetto formativo.
UploadProgettoFormativoET.jsp	Pagina per caricare il progetto formativo da parte dello Studente.
VisualizzaTirocinanteEnteET.jsp	Pagina per la visualizzazione delle richieste di Tirocinio da parte dell'Ente Convenzionato.
StoricoProprioTirocinioET.jsp	Pagina per la visualizzazione dello storico dei Tirocini da parte dello Studente.
StoricoStudenteET.jsp	Pagina per la visualizzazione dello storico dei Tirocini da parte della Segreteria.
firstForm.jsp	Pagina per la gestione della richiesta per “English Validation”.
uploadAttached.jsp	Pagina per la gestione dei file da caricare per “English Validation”.



3. Class interfaces

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

Nome Classe	EnteConvenzionato
Descrizione	Questa classe rappresenta le informazioni relative all' Ente Convenzionato.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome Classe	EnteConvenzionatoDAO
Descrizione	Questa classe definisce i metodi per interrogare, inserire, aggiornare ed eliminare i dati persistenti dell'Ente Convenzionato.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome Classe	Secretary
Descrizione	Questa classe rappresenta le informazioni relative alla Segreteria.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome Classe	Student
--------------------	---------



Descrizione	Questa classe rappresenta le informazioni relative allo Studente.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome Classe	Tirocinante
Descrizione	Questa classe rappresenta le informazioni relative al Tirocinante.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome Classe	TirocinanteDAO
Descrizione	Questa classe definisce i metodi per interrogare, inserire, aggiornare ed eliminare i dati persistenti del Tirocinante.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome Classe	Tirocinio
Descrizione	Questa classe rappresenta le informazioni relative al Tirocinio.
Pre-condizione	-
Post-condizione	-
Invarianti	-

Nome Classe	TirocinioDAO
Descrizione	Questa classe definisce i metodi per interrogare, inserire, aggiornare ed eliminare i dati persistenti del Tirocinio.
Pre-condizione	-



Post-condizione	-
Invarianti	-

Nome Classe	CheckSession
Descrizione	Questa classe gestisce le operazioni relative alla verifica della sessione Utente.
Pre-condizione	context CheckSession:: doPost(request:HttpServletRequest,response:HttpServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("user")!=null
Post-condizione	-
Invarianti	-

Nome Classe	DBConnection
Descrizione	Questa classe crea la connessione con il DB.
Pre-condizione	-
Post-condizione	context DBConnection:: createDBConnection(): Connection post: result!=null
Invarianti	-

Nome Classe	ServletAnnullaEnteDaStudenteET
Descrizione	Questa classe gestisce l'annullamento della richiesta all'Ente Convenzionato scelto precedentemente da parte dello Studente.
Pre-condizione	context ServletAnnullaEnteDaStudenteET:: do post(request:HttpServletRequest,response:HttpServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("userET").equals("0")
Post-condizione	context ServletAnnullaEnteDaStudenteET:: doPost(request:HttpServletRequest,response:HttpServletResponse)



	pServletResponse) post: Tirocinio.getStato().equals("Rifiutato")
Invarianti	-

Nome Classe	ServletAnnullaTirocinioET
Descrizione	Questa classe gestisce l'annullamento del Tirocinio da parte della Segreteria per un Tirocinante.
Pre-condizione	context ServletAnnullaTirocinioET:: do post(request:HttpServletRequest,response:Http pServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("userET").equals("1")
Post-condizione	context ServletAnnullaTirocinioET:: doPost(request:HttpServletRequest,response:Http pServletResponse) post: Tirocinio.getStato().equals("Annullato")
Invarianti	-

Nome Classe	ServletDocumentiTirocinioET
Descrizione	Questa classe gestisce la lista dei documenti da firmare.
Pre-condizione	context ServletDocumentiTirocinioET:: do post(request:HttpServletRequest,response:Http pServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("userET").equals("1") request.getSession().getAttribute("userET").equals("2") request.getSession().getAttribute("userET").equals("3") request.getSession().getAttribute("userET").equals("0")
Post-condizione	context ServletDocumentiTirocinioET:: doPost(request:HttpServletRequest,response:Http pServletResponse) post: ArrayList<Tirocinio> listaTirocinio.size()>=0
Invarianti	-

Nome Classe	ServletGestioneRichiesteEnteET
Descrizione	Questa classe gestisce l'accettazione ed il rifiuto delle richieste dall'Ente verso lo Studente.



Pre-condizione	context ServletGestioneRichiesteEnteET:: do post(request:HttpServletRequest,response:Http pServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("userET").equals("3")
Post-condizione	context ServletGestioneRichiesteEnteET:: doPost(request:HttpServletRequest,response:Http pServletResponse) post: (Tirocinio.getStatoTirocinio().equals("Accettato e in attesa di firma") && request.setAttribute("cod","1")) ((Tirocinio.getStatoTirocinio().equals("Rifiutato") && request.setAttribute("cod","3"))
Invarianti	-

Nome Classe	ServletGestioneRichiesteSegreteriaET
Descrizione	Questa classe gestisce l'accettazione ed il rifiuto delle richieste dalla Segreteria verso lo Studente.
Pre-condizione	context ServletGestioneRichiesteSegreteriaET:: do post(request:HttpServletRequest,response:Http pServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("userET").equals("1") context ServletGestioneRichiesteSegreteriaET:: do get(request:HttpServletRequest,response:Http pServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("userET").equals("1")
Post-condizione	(context ServletGestioneRichiesteSegreteriaET:: doPost(request:HttpServletRequest,response:Http pServletResponse) post: (Tirocinio.getStatoTirocinio().equals("in Attesa Ente") (Tirocinio.getStatoTirocinio().equals("Annullato"))) (context ServletGestioneRichiesteSegreteriaET::



	doGet(request:HttpServletRequest,response:HttpServletResponse) post: : ArrayList<Tirocinio> listaTirocini.size()>=0)
Invarianti	-

Nome Classe	ServletRichiestaInizioTirocinioET
Descrizione	Questa classe gestisce l'invio della richiesta di Inizio Tirocinio dallo Studente, inviata alla Segreteria.
Pre-condizione	context ServletRichiestaInizioTirocinioET:: do post(request:HttpServletRequest,response:HttpServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("userET").equals("0") && request.getSession().getAttribute("user")!= null)
Post-condizione	context ServletRichiestaInizioTirocinioET:: doPost(request:HttpServletRequest,response:HttpServletResponse) post: (Tirocinio.getStatoTirocinio().equals("In Attesta di Segreteria") && Tirocinante != null && Tirocinio != null
Invarianti	-

Nome Classe	ServletSceltaEnteET
Descrizione	Questa classe gestisce l'invio della richiesta all'Ente scelto dallo Studente.
Pre-condizione	context ServletSceltaEnteET:: do post(request:HttpServletRequest,response:HttpServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("userET").equals("0")
Post-condizione	context ServletSceltaEnteET:: doPost(request:HttpServletRequest,response:HttpServletResponse) post: (Tirocinio.getStatoTirocinio().equals("In Attesta Ente") && Tirocinante != null && Tirocinio != null
Invarianti	-



Nome Classe	ServletStatoTirocinanteET
Descrizione	Questa classe gestisce lo stato del Tirocinio di un singolo Tirocinante.
Pre-condizione	context ServletStatoTirocinanteET:: do post(request:HttpServletRequest,response:Http pServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("user")!=null
Post-condizione	context ServletStatoTirocinanteET:: doPost(request:HttpServletRequest,response:Http pServletResponse) post: Tirocinio!=null
Invarianti	-

Nome Classe	ServletStatoTirocinioET
Descrizione	Questa classe gestisce lo stato del Tirocinio di un singolo Tirocinante per la Segreteria.
Pre-condizione	context ServletStatoTirocinioET:: do post(request:HttpServletRequest,response:Http pServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("userET").equals("1")
Post-condizione	context ServletStatoTirocinanteET:: doPost(request:HttpServletRequest,response:Http pServletResponse) post: Tirocinio!=null && Tirocinante!= null
Invarianti	-

Nome Classe	ServletVisualizzaListaTirocinantiET
Descrizione	Questa classe gestisce la lista dei Tirocinanti per la Segreteria.
Pre-condizione	context ServletVisualizzaTirocinioET:: do post(request:HttpServletRequest,response:Http pServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("userET").equals("1")



Post-condizione	context <u>ServletVisualizzaTirocinioET::</u> doPost(request:HttpServletRequest,response:HttpServletResponse) post: ArrayList<Tirocinio> listaTirocinanti.size()>=0
Invarianti	-

Nome Classe	ServletCommon
Descrizione	Questa classe gestisce l'autenticazione dell'Utente.
Pre-condizione	context ServletVisualizzaTirocinioET:: do post(request:HttpServletRequest,response:HttpServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("flag").equals("1")
Post-condizione	context <u>ServletVisualizzaTirocinioET::</u> doPost(request:HttpServletRequest,response:HttpServletResponse) post: user != null && userET != null
Invarianti	-

Nome Classe	ServletListaEnteET
Descrizione	Questa classe gestisce la lista degli Enti Convenzionati.
Pre-condizione	context ServletListaEnteET:: do post(request:HttpServletRequest,response:HttpServletResponse) pre: request!=null && response!=null &&
Post-condizione	context <u>ServletListaEnteET::</u> doPost(request:HttpServletRequest,response:HttpServletResponse) post: ArrayList<EnteConvenzionato> listaEnti.size()>=0
Invarianti	-

Nome Classe	ServletRegistrazioneEnteET
Descrizione	Questa classe gestisce la registrazione di un nuovo Ente Convenzionato da parte della Segreteria.
Pre-condizione	context ServletRegistrazioneEnteET:: do post(request:HttpServletRequest,response:HttpServletResponse) pre: request!=null &&



	response!=null && request.getSession().getAttribute("userET").equals("1")
Post-condizione	context <u>ServletRegistrazioneEnteET</u> :: doPost(request:HttpServletRequest,response:HttpServletResponse) post : EnteConvenzionato !=null
Invarianti	-

Nome Classe	ServletModificaEnteET
Descrizione	Questa classe gestisce la modifica di un Ente Convenzionato da parte della Segreteria.
Pre-condizione	context ServletModificaEnteET:: do post(request:HttpServletRequest,response:HttpServletResponse) pre : request!=null && request.getSession().getAttribute("userET").equals("1") && response!=null && request.getParameter("Ente")!=null
Post-condizione	context ServletModificaEnteET:: doPost(request:HttpServletRequest,response:HttpServletResponse) post : EnteConvenzionato !=null
Invarianti	-

Nome Classe	ServletEliminaEnteET
Descrizione	Questa classe gestisce l'Eliminazione di un Ente non più Convenzionato da parte della Segreteria.
Pre-condizione	context ServletEliminaEnteET:: do post(request:HttpServletRequest,response:HttpServletResponse) pre : request!=null && request.getSession().getAttribute("userET").equals("1") && response!=null && request.getParameter("Ente")!=null
Post-condizione	context <u>ServletEliminaEnteET</u> :: doPost(request:HttpServletRequest,response:HttpServletResponse) post : response.setStatus (HttpServletResponse.SC_OK)
Invarianti	-

Nome Classe	ServletProgettoFormativoET
--------------------	----------------------------



Descrizione	Questa classe gestisce l'Upload ed il Download del "Progetto Formativo".
Pre-condizione	context ServletProgettoFormativoET:: do post(request:HttpServletRequest,response:Http pServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("user")!= null
Post-condizione	-
Invarianti	-

Nome Classe	ServletRecuperoPasswordET
Descrizione	Questa classe gestisce per il recupero password dell'Utente.
Pre-condizione	context ServletRecuperoPasswordET:: do post(request:HttpServletRequest,response:Http pServletResponse) pre: request!=null && response!=null && doRetriveByEmail()!=null
Post-condizione	context <u>ServletRecuperoPasswordET</u> :: doPost(request:HttpServletRequest,response:Http pServletResponse) post: request.getAttribute("Password","Modificata")
Invarianti	-

Nome Classe	ServletVisualizzaTirocinanteET
Descrizione	Questa classe gestisce la lista dei Tirocinanti da parte della Segreteria.
Pre-condizione	context ServletVisualizzataTirocinanteET:: do post(request:HttpServletRequest,response:Http pServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("userET").equals("1")
Post-condizione	context <u>ServletVisualizzataTirocinanteET</u> :: doPost(request:HttpServletRequest,response:Http pServletResponse) post: Tirocinio!= null && Tirocinante != null
Invarianti	-



Nome Classe	ServletVisualizzaTirocinanteEnteET
Descrizione	Questa classe gestisce la lista dei Tirocinanti da parte dell'EnteConvenzionato.
Pre-condizione	context ServletVisualizzataTirocinanteET:: do post(request:HttpServletRequest,response:HttpServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("userET").equals("3")
Post-condizione	context ServletVisualizzataTirocinanteET:: doPost(request:HttpServletRequest,response:HttpServletResponse) post: Tirocinio!= null && Tirocinante != null
Invarianti	-

Nome Classe	ServletUploadET
Descrizione	Questa classe gestisce l'upload del "Progetto Formativo".
Pre-condizione	context ServletVisualizzataTirocinanteET:: do post(request:HttpServletRequest,response:HttpServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("file")!=null
Post-condizione	context ServletVisualizzataTirocinanteET:: doPost(request:HttpServletRequest,response:HttpServletResponse) post: tirocinioDAO.uploadProgettoFormativo(tirocinio.getCodTirocinio(), pdfPath)==true
Invarianti	-

Nome Classe	ServletDownloadET
Descrizione	Questa classe gestisce il download del "Progetto Formativo".
Pre-condizione	context ServletVisualizzataTirocinanteET:: do post(request:HttpServletRequest,response:HttpServletResponse) pre: request!=null && response!=null && request.getSession().getAttribute("Tirocinio")!= null
Post-condizione	-

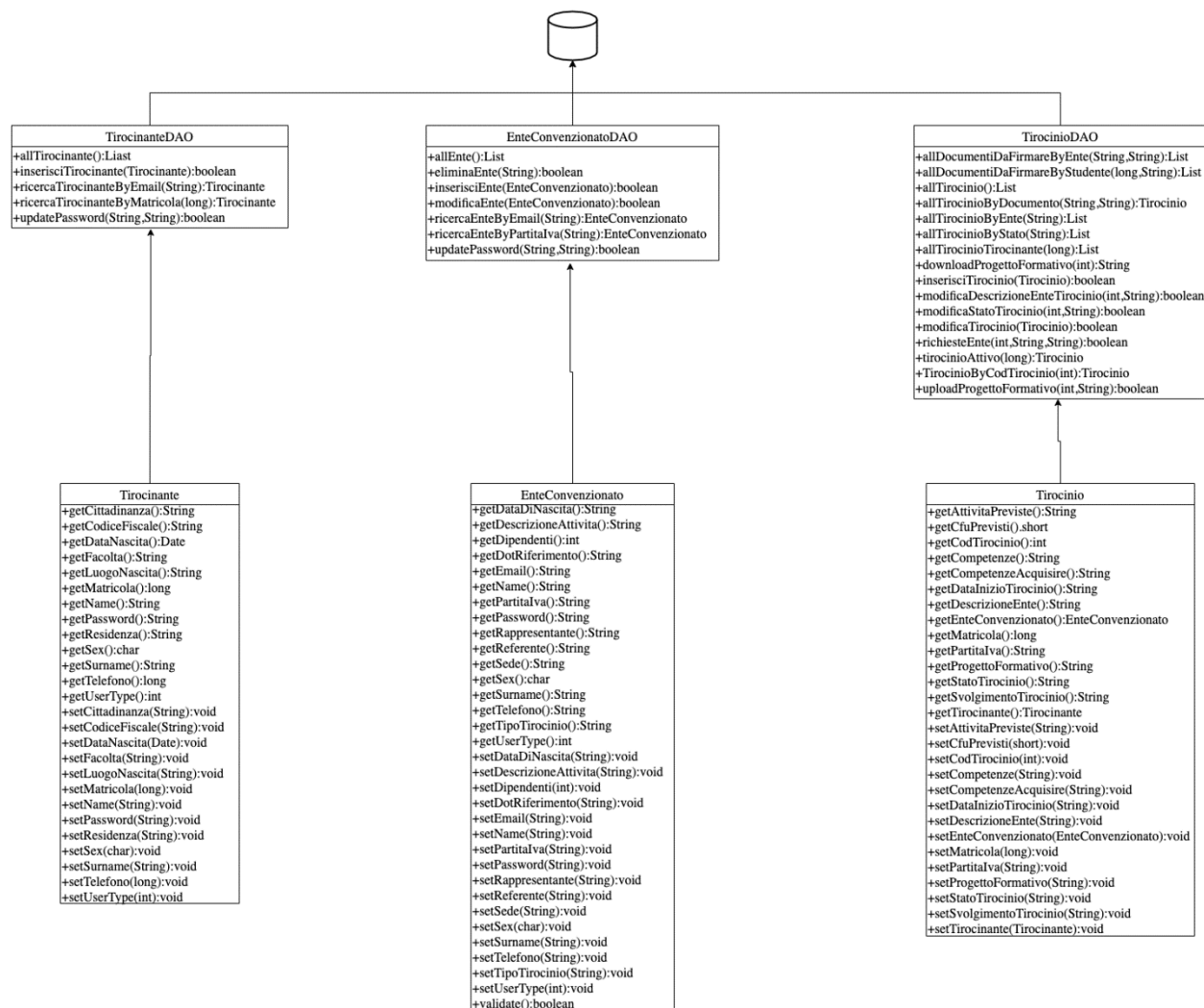
Invarianti

4. Design Pattern con Class Diagram

4.1. Design Pattern

4.1.1. Design Pattern DAO (Data Access Objects)

Il modello DAO è un modello strutturale che ci consente di isolare il livello applicazione, con relativa implementazione, dal livello di persistenza.



Il Design Pattern DAO ha come obiettivo:

- Separare le operazioni di accesso alla Base di Dati, dalle operazioni di business.



- Avere dell'interfaccia di riferimento, per l'accesso ai dati, per ogni classe JAVA implementata.

Utilizzo

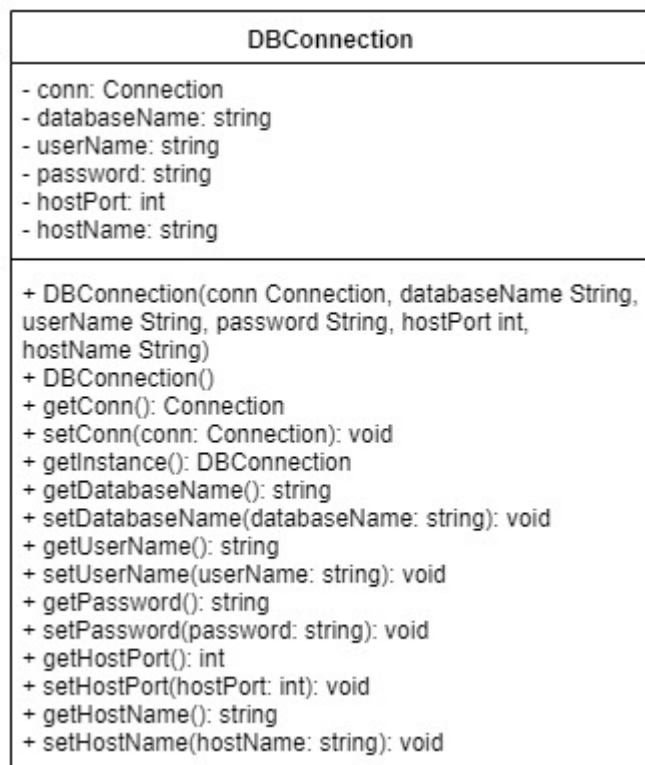
Il design pattern DAO sarà utilizzato per gestire le operazioni di persistenza, aggiornamento e rimozione dei dati, da effettuare in relazione con il Database.

Questo Design Pattern verrà utilizzato in concorrenza al modello MVC, di fatti nel package MODEL avremo la corrispondenza fra i DAO delle classi JAVA implementate e le varie entità all'interno del Database.

4.1.2. *Design Pattern Singleton*

Il Singleton è un Design Pattern Strutturale che ha lo scopo di garantire che venga creata una ed una sola istanza di una determinata classe e di fornire un punto di accesso globale a tale istanza, all'interno del Sistema.

Lo scope dell'oggetto istanziato è di applicazione e supporta la gestione degli accessi concorrenti ai metodi esposti.



Il Design Pattern Singleton ha come scopo:

- Avere un accesso controllato all'unica istanza della classe



- Ridurre il numero di oggetti condivisi
- Centralizzare informazioni e comportamenti in un'unica entità condivisa dagli utilizzatori

Il principale vantaggio offerto dall'uso di questo Design Pattern è la Mutua Esclusione.

Utilizzo

Poiché il Sistema è un'estensione del Sistema già esistente "*English Validation*" verrà utilizzata una singola classe (DBConnection) che consenta di effettuare tutte le operazioni con il Database.

Per evitare la perdita di efficienza dovuta alla creazione di più istanze di questa classe si è deciso di renderla un Singleton.

4.2. Class Diagram

Riferimento al documento: "*C2_ClassDiagram_ODD_Vers.1.0.png*"

5. Glossario

A

Admin: Un tipo di utente registrato al sistema che raffigura il Consiglio Didattico del dipartimento di Informatica degli Studi di Salerno.

Annullamento Tirocinio: nel sistema proposto è inteso come l'operazione che termina l'esperienza di Tirocinio Curriculare esterno dello studente per una sua personale scelta. Tale operazione può essere eseguita dalla Segreteria o dall'Admin.

C

Class Diagram: consente di descrivere tipi di entità, con le loro caratteristiche e le eventuali relazioni fra questi tipi.

Client: componente che accede a servizi e risorse di un altro componente detto server.

Componenti off-the-shelf: si riferisce a componenti hardware e software.

D

DAO: è un pattern architetturale per la gestione della persistenza.

DBMS: sistema software per creazione, manipolazione e interrogazione efficiente di database.

Design Pattern: una soluzione progettuale generale ad un problema ricorrente.

Dipartimento: Il Dipartimento di informatica degli Studi di Salerno.

E

Ente Convenzionato: L'ente/azienda preposta al Tirocinio Esterno.



EV: Il Sistema "*English Validation*" sviluppato per la convalida dei CFU per la lingua inglese.

H

HTTP: protocollo di trasferimento di ipertesti che consente a due macchine, client e server, di interagire attraverso un meccanismo di richiesta/risposta. Il client inoltra una richiesta al server, che verrà soddisfatta con la risposta di quest'ultimo.

J

JSP: tecnologia di programmazione web utilizzata per fornire contenuti dinamici.

M

Mutua Esclusione: procedimento di sincronizzazione fra processi o thread concorrenti, con il quale si impedisce che più task paralleli accedano contemporaneamente ai dati in memoria o ad altre risorse soggette a race condition.

MVC: Architettura composta da Model, il quale fornisce i metodi per accedere ai dati utili all'applicazione, View, visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti, e Controller, riceve i comandi dell'utente e li attua modificando lo stato degli altri due componenti.

N

NRF: Requisito non funzionale.

O

ODD: Il documento di Object Design chiude il gap tra oggetti di applicazione e componenti off-the-shelf identificando oggetti di soluzione e raffinando gli oggetti esistenti.

P

Package: collezione di classi e interfacce correlate.

Progetto Formativo: documento che contiene tutte le informazioni riguardanti un tirocinio.

R

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

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

Rifiuto: tutte le richieste effettuabili possono essere rifiutate. Il rifiuto è solitamente una conseguenza di una mancanza di requisito oppure di dati incongruenti, in questo caso studente è tenuto a ripetere la richiesta. Tale operazione può essere estesa a più attori: Ente, Segreteria ed Admin.



S

Script: insieme di strumenti per la programmazione, più facili da utilizzare rispetto ai tradizionali linguaggi.

SDD: Il documento del System Design è un'attività preliminare allo sviluppo di un sistema software. Gli scopi del documento sono quelli di definire gli obiettivi di progettazione del sistema e di decomporre il sistema in sottosistemi più piccoli.

Segreteria: Tale figura svolge il ruolo di amministrazione didattica nell'ambito del dipartimento d'appartenenza. Essa acquisisce una valenza significativa e nel sistema corrente ed in quello proposto.

Server: componente che gestisce traffico di informazioni e fornisce servizi e risorse attraverso la rete.

Servlet: oggetti Java all'interno del server web che permettono di creare web applications in combinazione con JSP.

Shutdown: spegnimento di un sistema

Singleton: è un design pattern creazionale che ha lo scopo di garantire che di una determinata classe venga creata una e una sola istanza e di fornire un punto di accesso globale a tale istanza.

Sottosistema: un sistema subalterno e secondario rispetto a un sistema più complesso.

Startup: avvio di un sistema.

T

Tirocinio: Tirocinio Curriculare Esterno dedicato agli studenti del corso di studi del Dipartimento di Informatica.

Trade-off: è una situazione che implica una scelta tra due o più possibilità, in cui la perdita di valore di una costituisce un aumento di valore in un'altra.

U

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

UNISA: Università degli Studi di Salerno.

Utente: Un qualsiasi utilizzatore della piattaforma.

W

Web Browser: applicazione software installata sul client che permette di visualizzare e navigare le risorse del web.



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