ODD
Object Design Document

Riferimento	
Versione	1.0
Data	16/12/2018
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Mario De Cicco, Mario Greco, Giovanni Di Nocera, Anna Maria Raffaela Riccio

Revision History

Data	Versione	Descrizione	Autori
06/12/2018	0.01	Layout del documento	Tutti
08/12/2018	0.02	Trade off e componenti off the shelf	Tutti
10/12/2018	0.03	Class interfaces	Tutti
11/12/2018	0.04	Linee Guida, definizione acronimi	Tutti
12/12/2018	0.04	Revisione layout documento	Tutti
14/12/2018	0.05	Class Interface	Tutti
16/12/2018	0.06	Class Diagram e Glossario	Tutti

Sommario

1. Introduzione

1.1 Object design Trade-offs

1.1.1 Comprensibilità vs Costi
Si preferisce aggiungere costi relativi alle ore/uomo per la documentazione, al fine di rendere il codice comprensibile sia alle persone attualmente non coinvolte nel progetto che alle persone coinvolte. Saranno introdotti commenti nel codice che ne facilitano la comprensione e, di conseguenza agevolano la manutenzione.

1.1.2 Prestazioni vs Costi
Il budget a disposizione non è eccessivo, perciò il sistema sarà sviluppato utilizzando componenti open source e free. Non saranno garantite alte prestazioni, che saranno comunque soddisfacenti per il normale utilizzo del sistema.

1.1.3 Comprare vs Costruire
Si preferisce comprare in quanto i componenti off-the-shelf utilizzate sono open source e quindi non richiedono spese aggiuntive.

1.1.4 Costi vs Mantenimento
Il sistema sviluppato può essere facilmente modificato ed implementato con nuove funzioni e corretto in presenza di errori.

1.1.5 Interfaccia vs Easy-use
L'interfaccia permette un facile utilizzo delle principali funzionalità del sistema anche per gli utenti meno esperti.

1.1.6 Sicurezza vs Costi
Dato il budget ridotto, non saranno utilizzate componenti esterni che garantiscano la massima sicurezza sui dati, ma verrà utilizzata una componente all'interno del linguaggio java che permetta un grado di protezione soddisfacente.

1.1.7 Interfacce vs Tempo di risposta
Il tempo di risposta tra server e interfaccia è sufficiente a soddisfare le richieste da parte dell'utente.

1.2 Componenti off-the-shelf

Per il progetto software che si vuole realizzare sono utilizzate componenti off-the-shelf, cioè componenti software disponibili sul mercato per facilitare la creazione del progetto.

Per il sistema che si vuole realizzare ci interessano un framework ed una libreria JavaScript per applicazioni web: Bootstrap e JQuery. Il bootstrap è un framework open source che contiene una raccolta di strumenti liberi per la creazione di siti e applicazioni per il web. Questo contiene modelli di progettazione HTML e CSS, sia per la tipografia, che per le varie componenti interfaccia come moduli, bottoni, navigazione e altre componenti dell'interfaccia, così come alcune estensioni opzionali di JavaScript. Si basa esclusivamente su tecnologie native per il browser e non richiede plug-in lato client come Flash o Java.

JQuery è un framework JavaScript open source che rende più semplice manipolare il DOM delle pagine HTML.

1.3 Linee guida per la documentazione dell'interfaccia

Per avere una maggiore manutenibilità ed estensibilità del codice e una più efficiente comunicazione, prima dell'implementazione della logica del sistema, è opportuno sottomettere regole di implementazione, in modo che eventuali correzioni nella logica dell'applicazione possano essere apportate prima di imbattersi nella sintassi degli strumenti scelti.

1.3.1 Linee guida generali:

- A tutti i metodi, le classi e i file prodotti, deve essere allegato un commento che specifichi l'obiettivo da raggiungere e il perché delle decisioni prese
- Prima dell'implementazione della logica di un algoritmo, esso sarà rivisto da tutti i membri del team per la correzione di eventuali errori nella logica

1.3.2 Organizzazione di metodi, parametri, variabili e classi

- La convenzione "Notazione a Cammello" (scrivere parole composte o frasi unendo tutte le parole tra loro ma lasciando le iniziali in maiuscolo) deve essere adottata da tutti i membri del team. Per i nomi delle classi si utilizza UpperCamelCase, mentre per i nomi dei metodi si utilizza lowerCamelCase
- I nomi dei pacchetti sono tutti in minuscolo, con parole consecutive semplicemente concatenate insieme (senza underscore)
- I nomi dei campi non costanti (statici o meno) sono scritti in lowerCamelCase
- I nomi delle variabili locali sono scritti in lowerCamelCase
- Un metodo è contrassegnato con l'annotazione @Override ogni volta che è necessario. Ciò include un metodo di classe che sostituisce un metodi di superclasse, un metodo che

implementa un metodo di interfaccia e un metodo di interfaccia che rispetta un metodo di superinterfaccia

- Le annotazioni che si applicano ad una classe, un metodo o un costruttore appaiono immediatamente dopo il blocco della documentazione ed ogni annotazione è elencata su una riga a sé stante. Queste interruzioni di riga non costituiscono il ritorno a capo, quindi il livello di indentazione non viene aumentato
- Ogni dichiarazione di variabile dichiara solo una variabile: Int a, b: non vengono utilizzate
- Al di la di dove richiesto dalla lingua o altre regole di stile, un unico spazio è inserito nei seguenti blocchi:
    - I due punti (:) in una istruzione avanzata FOR ("foreach")
    - Tra il tipo e la variabile di una dichiarazione : List<String> list
- I file di origine sono codificati in UTF-8
- Le classi avranno il nome al singolare e l'estensione ".java"
- I nomi dei campi e dei parametri saranno dei sostantivi mentre, quelli dei metodi saranno dei verbi
- Nel caso venga usato più volte lo stesso valore numerico all'interno del codice, è opportuno inizializzare una costante con tale valore
- Inizializzare le variabili locali nel punto in cui sono state dichiarate a meno che il suo valore iniziale non dipenda da un calcolo che occorre eseguire prima

1.3.3 Organizzazione dei file

- Ogni file deve essere sviluppato e diviso in base alla categoria di appartenenza, ovvero deve essere correlato ad un'unica funzionalità che persegue
- L'importazione statica non viene utilizzata per le classi nidificate statiche. Sono importati con normali importazioni
- Ogni classe di livello superiore risiede in un proprio file sorgente
- Ogni classe utilizza un ordine logico, ad esempio i nuovi metodi non vengono aggiunti di solito alla fine della classe, poiché ciò restituirebbe un ordinamento cronologico per data di aggiunta, che non è un ordinamento logico
- Quando una classe ha più costruttori o più metodi con lo stesso nome, questi appaiono in sequenza, senza altri codici
- Per i nomi dei file, delle operazioni e delle variabili verranno utilizzati nomi evocativi ed in lingua italiana. Le uniche eccezioni saranno GET e SET, inseriti nella nostra terminologia data la familiarità con il linguaggio di programmazione Java e perchè anche la migliore traduzione non regge il confronto
- Organizzare in una cartella il file delle librerie usate e le altre risorse scaricate necessarie per lo sviluppo del progetto

1.3.4 Indentazione

- L'indentazione deve essere effettuata con un TAB e qualunque sia il linguaggio usato per la produzione del codice, ogni soluzione deve essere opportunamente indentata

<html>

```
        <head>
        </head>
        <body>
        </body>
    <html>
```

Questa pratica deve essere usata soprattutto per le istruzioni FOR e IF.

- Ogni volta che viene aperto un nuovo blocco o un costrutto simile ad un blocco, il rientro aumenta di uno spazio. Al termine del blocco, il rientro torna al livello di rientro precedente
- Ogni dichiarazione è seguita da un'interruzione di riga
- È buona pratica scendere di livello

1.3.5 Script JavaScript

- Gli script che svolgono funzioni diverse dal funzionamento di una pagina, dovrebbero essere collocati in file separati
- Le funzioni e oggetti in JavaScript devono essere preceduti da un commento in stile JavaDoc

1.3.6 Posizione

- Mettere le dichiarazioni all'inizio dei blocchi
- Non aspettare di dichiarare le variabili al loro primo uso, poiché questo potrebbe confondere il programmatore inesperto, impedire la portabilità del codice dentro lo Scope. L'unica eccezione a questa regola sono gli indici dei cicli FOR che in Java possono essere dichiarati nell'istruzione stessa
- Evitare dichiarazioni locali che nascondono dichiarazioni a più alto livello, ad esempio non dichiarare una variabile con lo stesso nome in un blocco intero

1.3.7 Parentesi

- A prescindere dalle istruzioni che seguono un IF, ELSE, FOR, DO, WHILE, è necessario, laddove ci fosse anche una sola istruzione, riportare il blocco di istruzioni tra parentesi graffe
- Ogni TAG di apertura deve essere seguito dall'apposito TAG di chiusura (eccetto i TAG self-closing)

1.4 Design pattern

Façade pattern: si è deciso di utilizzare il Façade pattern per definire un'unica interfaccia a livello di logica che permette all'utente di interagire, attraverso l'interfaccia, con le funzionalità della piattaforma vedendole come un unico sistema.

DAO pattern: si è deciso di utilizzare il DAO pattern per stratificare e isolare l'accesso ad una tabella tramite query, ovvero al data layer da parte della business logic creando un maggiore livello di

astrazione ed una più facile manutenibilità. I metodi del DAO con le rispettive query dentro verranno così richiamati dalle classi della business logic

1.5 Definizioni, acronimi e abbreviazioni

DBMS: Data Base Management System

Off-The-Shelf: Servizi esterni di cui viene fatto utilizzo da terzi

JQuery: È una libreria JavaScript open source per applicazioni web. Nasce con l'obiettivo di semplificare la selezione, la manipolazione, la gestione degli eventi e l'animazione di elementi DOM in pagine HTML, oltre ad implementare funzionalità AJAX.

Framework: Software di supporto allo sviluppo web

Java Doc: È un applicativo incluso nel Java Development Kit utilizzato per la generazione automatica della documentazione del codice sorgente scritto in linguaggio Java

HTML: Linguaggio di mark-up per pagine web

Bootstrap: Framework che contiene librerie utili per lo sviluppo responsive di pagine web

Eclipse: Ambiente di sviluppo integrato multi-linguaggio e multi-piattaforma

JavaScript: Linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione, in siti web e applicazioni web, di effetti dinamici interattivi tramite funzioni di script invocate da eventi innescati a loro volta in vari modi dall'utente sulla pagina web in uso

CSS: Linguaggio usato per definire la formattazione di pagine web.

1.6 Riferimenti

- TF_RAD_v3.0
- SDD_v2.0
- Bernd Bruegge & Allen H. Dutoit, Object-Oriented Software Engineering: Using UML, Patterns and Java, (3nd edition), Prentice-Hall, 2009.
- https://getbootstrap.com/
- https://jquery.com/

2. Package

Packages	
Nome	**Descrizione**
WebContent	Contiene tutte le classi jsp che vengono mostrare all'utente per inserire o visualizzare dati, queste classi fanno richiesta alle servlet presenti nel package it.tirociniofast.control e contiene la cartella img dove sono memorizzate le immagini presenti nelle pagine della piattaforma.
it.tirociniofast.control	Contiene tutte le classi servlet che prendono i parametri dalle jsp e eseguono i metodi logici chiamando le classi model presenti nel package it.tirociniofast.model
it.tirociniofast.model	Contiene tutte le classi model che eseguono le operazioni CRUD sul database, istanziando oggetti bean presenti nel package it.tirociniofast.bean e stabilendo una connessione al DB con gli oggetti del package it.tirociniofast.storage
it.tirociniofast.bean	Contiene tutte le classi bean che rappresentato i dati persistenti
javax.Sql.DataSource	Contiene tutte le classi per stabilire una connessione con il database MySql

3. Interfaccia delle classi

Nome classe	GestioneUtente
Descrizione	Servlet che rappresenta le funzionalità relative alla gestione del login, logout, recupera password, imposta password, registrazione, visualizzazione area personale, compilazione e completamento della scheda di un'azienda. • login(HttpServletRequest request, HttpServletResponse response) • logout(HttpServletRequest request, HttpServletResponse response) • recuperaPassword(HttpServletRequest request, HttpServletResponse response)

	• impostaPassword(HttpServletRequest request, HttpServletResponse response) • registrazione(HttpServletRequest request, HttpServletResponse response) • visualizzazioneAreaPersonale(HttpServletRequest request, HttpServletResponse response) • compilazione(HttpServletRequest request, HttpServletResponse response) • completamentoSchedaAzienda(HttpServletRequest request, HttpServletResponse response)
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	GestioneTirocinio
Descrizione	Servlet che rappresenta le funzionalità relative alla gestione dell'inoltro di una richiesta di attività di tirocinio, della visualizzazione dell'elenco delle aziende convenzionate, della visualizzazione della scheda di un'azienda e della ricerca di un'azienda. • inoltraRichiestaTirocinio(HttpServletRequest request, HttpServletResponse response) • visualizzaElencoAziende(HttpServletRequest request, HttpServletResponse response) • visualizzaSchedaAzienda(HttpServletRequest request, HttpServletResponse response) • ricercaAzienda(HttpServletRequest request, HttpServletResponse response)
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	GestioneConvenzione
Descrizione	Servlet che rappresenta le funzionalità relative alla gestione dell'inoltro di una convenzione. • inoltraConvenzione(HttpServletRequest request, HttpServletResponse response)
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	GestioneQuestionario

Descrizione	Servlet che rappresenta le funzionalità relative alla gestione dell'inoltro di un quastionario valutativo. • inoltraQuestionario(HttpServletRequest request, HttpServletResponse response)
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	GestioneDocumento
Descrizione	Servlet che rappresenta le funzionalità relative alla gestione della compilazione, visualizzazione, download, upload, convalida di un documento, dell'inoltro del registro ore, della visualizzazione dell'elenco dei documenti, quali richieste di attività di tirocinio e convenzioni Azienda-Università. • compilazione(HttpServletRequest request, HttpServletResponse response) • download(HttpServletRequest request, HttpServletResponse response) • upload(HttpServletRequest request, HttpServletResponse response) • convalidaDocumento(HttpServletRequest request, HttpServletResponse response) • inoltroRegistroOre(HttpServletRequest request, HttpServletResponse response) • visualizzaElencoRichiesteTirocino(HttpServletRequest request, HttpServletResponse response) • visualizzaElencoConvenzioni(HttpServletRequest request, HttpServletResponse response)
Pre-Condizione	
Post-Condizione	
Invarianti	

Nome classe	UtenteModel
Descrizione	Rappresenta i metodi per accedere ai dati relativi agli utenti, modificarli e salvarli. • recuperaDatiUtente(String username, String password) • recuperaPassword(String username, String risposta) • impostaPassword(String password, String confermaPassword) • registraStudente(String nome, String cognome, String luogoNascita, String dataNascita, String indirizzo, String residenza, String codiceFiscale, int matricola, String email, String telefono, String username, String password, String confermaPassword, String risposta) • registraAzienda(String nome, int partitaIVA, String CEO, String indirizzo, String email, String telefono, String username, String password, String confermaPassword, String risposta)

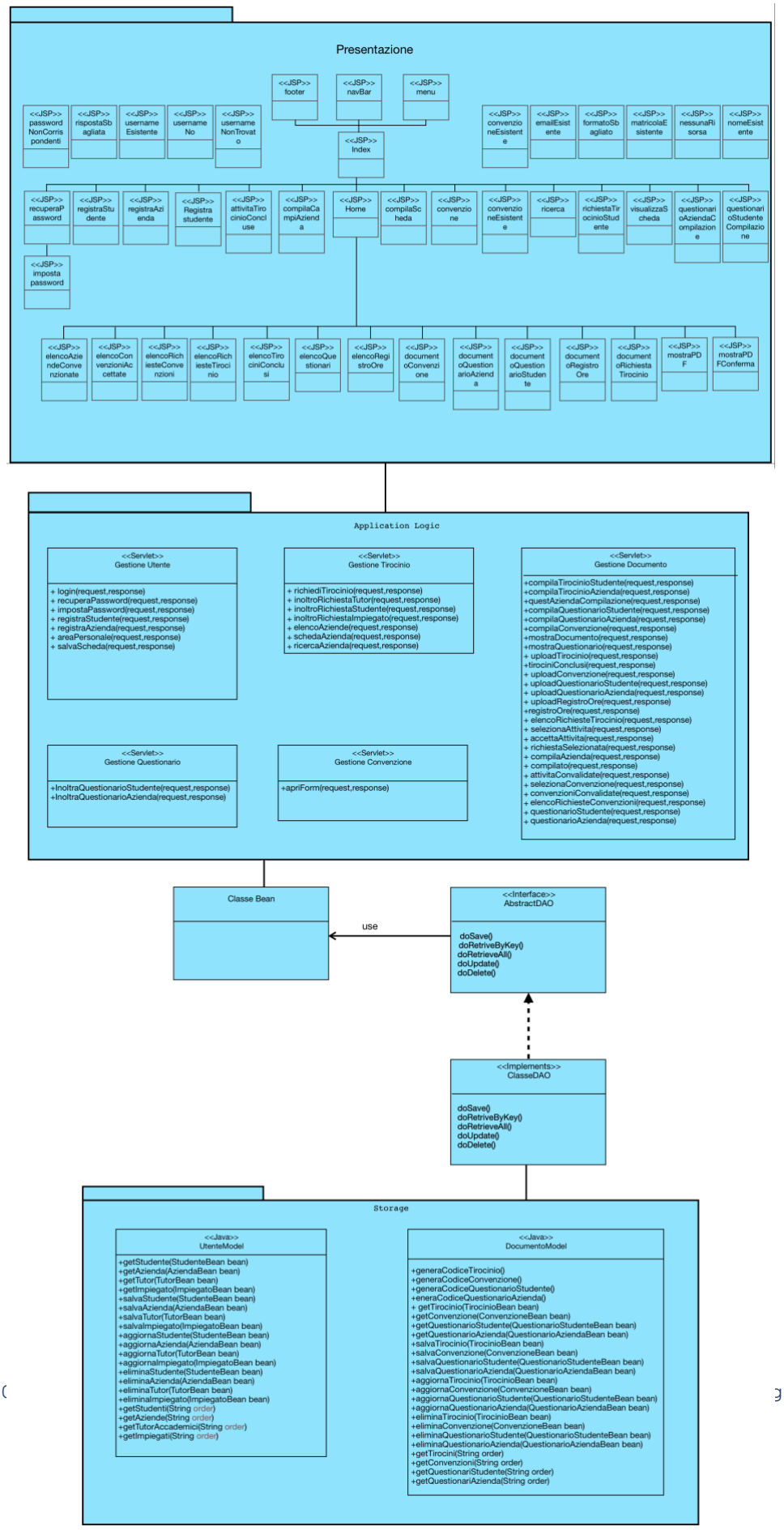
	• completaSchedaAzienda(int idAzienda, String pathImg, String descrizione)
Pre-Condizione	**context** UtenteModel :: recuperaDatiUtente(String username, String password); **pre:** username != null && password != null **context** UtenteModel :: recuperaPassword(String username, String risposta); **pre:** username != null && risposta != null **context** UtenteModel:: impostaPassword(String password, String confermaPassword); **pre:** password != null && confermaPassword != null **context** UtenteModel:: registraStudente(String nome, String cognome, String luogoNascita, String dataNascita, String indirizzo, String residenza, String codiceFiscale, int matricola, String email, String telefono, String username, String password, String confermaPassword, String risposta); **pre**: nome != null, cognome != null, luogoNascita != null, dataNascita != null, indirizzo != null, residenza != null, codiceFiscale != null, matricola != null, email != null, telefono != null, username != null, password != null, confermaPassword != null, risposta != null **context** UtenteModel:: registraAzienda (String nome, int partitaIVA, String CEO, String indirizzo, String email, String telefono, String username, String password, String confermaPassword, String risposta) **pre:** nome != null, partitaIVA != null, CEO != null, indirizzo != null, email != null, telefono != null, username != null, password != null, confermaPassword != null, risposta != null **context** UtenteModel: completaSchedaAzienda(String pathImg, String descrizione) **pre:** idAzienda != null, pathImg !=null && descrizione != null
Post-Condizione	
Invarianti	

Nome classe	TirocinioModel
Descrizione	Rappresenta i metodi per accedere ai dati relativi alle aziende. • ricercaAzienda(String nome, String sede) • ricercaAziendaNome(String nome) • ricercaAziendaSede(String sede)
Pre-Condizione	**context** TirocinioModel: ricercaAzienda(String nome, String sede) **pre:** nome != null && sede != null **context** TirocinioModel: ricercaAziendaNome(String nome) **pre:** nome != null **context** TirocinioModel: ricercaAziendSede(String sede) **pre:** sede != null

Post-Condizione	
Invarianti	

Nome classe	DocumentoModel
Descrizione	Rappresenta i metodi per accedere ai dati relativi ai documenti. • Download(String pathDoc) • Upload(String pathDoc) • recuperaRichiesteTirocinio() • recuperaConvenzioni() • compilaRichiestaTirocinioStudente(int idStudente, int idAzienda, int idTutorAccademico, String annoAccademico, int cfu, boolean handicap) • compilaRichiestaTirocinioAzienda(int idAzienda, int idStudente, String sede, String periodo, String tempiAccesso, String obiettiviTirocinio, String facilitazioni) • compilaRichiestaConvenzione(int idAzienda, String luogoNascita, String dataNascita, int numDipendenti, String referente, String telefonoRef, String emailRef, String descrizione, String docente) • compilaQuestionarioStudente(int idStudente, int idAzienda, int idTutorAccademico, String titolo, String periodo) • compilaQuestionarioAzienda(int idAzienda, int idStudente, int idTutorAccademico, String titolo, String periodo, String posizione) • convalidaTirocinio(int idStudente, int idAzienda, int idTutorAccademico, String pathRichiestaTirocinio, String pathQuestionarioStudente, String pathQuestionarioAzienda, String pathRegistroOre) • convalidaConvenzione(int idAzienda, String pathConvenzione)
Pre-Condizione	**context** DocumentoModel:: Download(pathDoc); **pre** pathDoc != null; **context** DocumentoModel:: Upload(idUtente, pathDoc, tipo); **pre** pathDoc != null && tipo != null; **context** DocumentoModel:: compilaRichiestaTirocinioStudente(idStudente, idAzienda, idTutorAccademico, annoAccademico, cfu, handicap); **pre** annoAccademico > 0 && cfu > 0 && handicap != null; **context** DocumentoModel:: compilaRichiestaTirocinioAzienda(idAzienda, idStudente, sede, periodo, tempiAccesso, obiettiviTirocinio, facilitazioni); **pre** sede != null && periodo != null && tempiAccesso != null && obiettiviTirocinio != null && facilitazioni != null; **context** DocumentoModel:: compilaRichiestaConvenzione(idAzienda, luogoNascita, dataNascita, numDipendenti, referente, telefonoRef, emailRef, descrizione, docente); **pre** luogoNascita != null && dataNascita != null && numDipendenti > 0 && referente != null && telefonoRef != null && emailRef != null && descrizione != null && docente != null; **context** compilaQuestionarioStudente(idStudente, idAzienda, idTutorAccademico, titolo, periodo); **pre** titolo != null && periodo != null; **context** compilaQuestionarioAzienda(idAzienda, idStudente, idTutorAccademico, titolo, periodo, posizione);

	pre titolo != null && periodo != null && posizione != null; **context** convalidaTirocinio(idStudente, idAzienda, idTutorAccademico, pathRichiestaTirocinio, pathQuestionarioStudente, pathQuestionarioAzienda, pathRegistroOre); **pre** pathRichiestaTirocinio != null && pathQuestionarioStudente != null && pathQuestionarioAzienda != null && pathRegistroOre != null; **context** convalidaConvenzione(idAzienda, pathConvenzione); **pre** pathConvenzione != null;
Post-Condizione	
Invarianti	


Presentazione
<<JSP>> footer
<<JSP>> navBar
<<JSP>> menu
<<JSP>> Index
<<JSP>> password NonCorrispondenti
<<JSP>> rispostaSbagliata
<<JSP>> username Esistente
<<JSP>> username No
<<JSP>> username NonTrovato
<<JSP>> convenzioneEsistente
<<JSP>> emailEsistente
<<JSP>> formatoSbagliato
<<JSP>> matricolaEsistente
<<JSP>> nessunaRisorsa
<<JSP>> nomeEsistente
<<JSP>> recuperaPassword
<<JSP>> registraStudente
<<JSP>> registraAzienda
<<JSP>> Registra studente
<<JSP>> attivitaTirocinioConclUse
<<JSP>> compilaCampiAzienda
<<JSP>> Home
<<JSP>> compilaScheda
<<JSP>> convenzione
<<JSP>> convenzioneEsistente
<<JSP>> ricerca
<<JSP>> richiestaTirocinioStudente
<<JSP>> visualizzaScheda
<<JSP>> questionariOAziendaCompilazione
<<JSP>> questionariOStudenteCompilazione
<<JSP>> imposta password
<<JSP>> elencoAziendeConvenzionate
<<JSP>> elencoConvenzioniAccettate
<<JSP>> elencoRichiesteConvenzioni
<<JSP>> elencoRichiesteTirocinio
<<JSP>> elencoTirociniConclusi
<<JSP>> elencoQuestionari
<<JSP>> elencoRegistroOre
<<JSP>> documentoConvenzione
<<JSP>> documentoQuestionarioAzienda
<<JSP>> documentoQuestionarioStudente
<<JSP>> documentoRegistroOre
<<JSP>> documentoRichiestaTirocinio
<<JSP>> mostraPDF
<<JSP>> mostraPDFConferma
Application Logic
<<Servlet>> Gestione Utente
+ login(request,response)
+ recuperaPassword(request,response)
+ impostaPassword(request,response)
+ registraStudente(request,response)
+ registraAzienda(request,response)
+ areaPersonale(request,response)
+ salvaScheda(request,response)
<<Servlet>> Gestione Tirocinio
+ richiediTirocinio(request,response)
+ inoltroRichiestaTutor(request,response)
+ inoltroRichiestaStudente(request,response)
+ inoltroRichiestaImpiegato(request,response)
+ elencoAziende(request,response)
+ schedaAzienda(request,response)
+ ricercaAzienda(request,response)
<<Servlet>> Gestione Documento
+compilaTirocinioStudente(request,response)
+compilaTirocinioAzienda(request,response)
+questAziendaCompilazione(request,response)
+compilaQuestionarioStudente(request,response)
+compilaQuestionarioAzienda(request,response)
+compilaConvenzione(request,response)
+mostraDocumento(request,response)
+mostraQuestionario(request,response)
+ uploadTirocinio(request,response)
+tirociniConclusi(request,response)
+ uploadConvenzione(request,response)
+ uploadQuestionarioStudente(request,response)
+ uploadQuestionarioAzienda(request,response)
+ uploadRegistroOre(request,response)
+registroOre(request,response)
+ elencoRichiesteTirocinio(request,response)
+ selezionaAttivita(request,response)
+ accettaAttivita(request,response)
+ richiestaSelezionata(request,response)
+ compilaAzienda(request,response)
+ compilato(request,response)
+ attivitaConvalidate(request,response)
+ selezionaConvenzione(request,response)
+ convenzioniConvalidate(request,response)
+ elencoRichiesteConvenzioni(request,response)
+ questionarioStudente(request,response)
+ questionarioAzienda(request,response)
<<Servlet>> Gestione Questionario
+InoltraQuestionarioStudente(request,response)
+InoltraQuestionarioAzienda(request,response)
<<Servlet>> Gestione Convenzione
+apriForm(request,response)
Classe Bean
use
<<Interface>> AbstractDAO
doSave()
doRetriveByKey()
doRetrieveAll()
doUpdate()
doDelete()
<<Implements>> ClasseDAO
doSave()
doRetriveByKey()
doRetrieveAll()
doUpdate()
doDelete()
Storage
<<Java>> UtenteModel
+getStudente(StudenteBean bean)
+getAzienda(AziendaBean bean)
+getTutor(TutorBean bean)
+getImpiegato(ImpiegatoBean bean)
+salvaStudente(StudenteBean bean)
+salvaAzienda(AziendaBean bean)
+salvaTutor(TutorBean bean)
+salvaImpiegato(ImpiegatoBean bean)
+aggiornaStudente(StudenteBean bean)
+aggiornaAzienda(AziendaBean bean)
+aggiornaTutor(TutorBean bean)
+aggiornaImpiegato(ImpiegatoBean bean)
+eliminaStudente(StudenteBean bean)
+eliminaAzienda(AziendaBean bean)
+eliminaTutor(TutorBean bean)
+eliminaImpiegato(ImpiegatoBean bean)
+getStudenti(String order)
+getAziende(String order)
+getTutorAccademici(String order)
+getImpiegati(String order)
<<Java>> DocumentoModel
+generaCodiceTirocinio()
+generaCodiceConvenzione()
+generaCodiceQuestionarioStudente()
+eneraCodiceQuestionarioAzienda()
+ getTirocinio(TirocinioBean bean)
+getConvenzione(ConvenzioneBean bean)
+getQuestionarioStudente(QuestionarioStudenteBean bean)
+getQuestionarioAzienda(QuestionarioAziendaBean bean)
+salvaTirocinio(TirocinioBean bean)
+salvaConvenzione(ConvenzioneBean bean)
+salvaQuestionarioStudente(QuestionarioStudenteBean bean)
+salvaQuestionarioAzienda(QuestionarioAziendaBean bean)
+aggiornaTirocinio(TirocinioBean bean)
+aggiornaConvenzione(ConvenzioneBean bean)
+aggiornaQuestionarioStudente(QuestionarioStudenteBean bean)
+aggiornaQuestionarioAzienda(QuestionarioAziendaBean bean)
+eliminaTirocinio(TirocinioBean bean)
+eliminaConvenzione(ConvenzioneBean bean)
+eliminaQuestionarioStudente(QuestionarioStudenteBean bean)
+eliminaQuestionarioAzienda(QuestionarioAziendaBean bean)
+getTirocini(String order)
+getConvenzioni(String order)
+getQuestionariStudente(String order)
+getQuestionariAzienda(String order)

4. Glossario

ODD (Object design Document): Documento che approfondisce l'analisi dei requisiti e rappresenta le decisioni relative all'implementazione.

Facade: Un oggetto che permette, attraverso un'unica interfaccia, l'accesso a sottosistemi che espongono interfacce complesse e molto diverse tra loro, nonché a blocchi di codice complessi.

Class interface: Sezione dell'ODD che descrive le classi definite per l'implementazione, indicando i contratti (formati da precondizioni, postcondizioni e invarianti).

Componenti off-the-shelf: componenti hardware e software già disponibili, open-source o meno, che possono essere usate all'interno di un progetto.

HTML (HyperText markup language): Linguaggio di markup per pagine web.

CSS (Cascading Style Sheets): Linguaggio usato per la formattazione di pagine HTML, XHTML E XML.

JQuery: Libreria JavaScript per la gestione di eventi e l'animazione di elementi DOM in una pagina HTML.

Bootstrap: Raccolta di software open-source per la creazione di siti e applicazioni web.

JavaScript: Linguaggio di programmazione orientato ad oggetti e ad eventi, utilizzato per la realizzazione di effetti dinamici interattivi in siti e applicazioni web.

Servlet: Oggetti in linguaggio Java utilizzati per la creazione di un'applicazione web.

Bean model: Modelli usati per rappresentare dei dati salvati permanentemente.

DAO: Design pattern che permette di stratificare e isolare l'accesso ad una tabella tramite query

Camel Notation: La pratica, tipica del linguaggio di programmazione Java, di scrivere le parole composte senza spazi, mettendo in maiuscolo le iniziali.

DBMS (Database management system): Sistema software utilizzato per la gestione di database.

AJAX (Asynchronus JavaScript and XML): Tecnica di sviluppo software di applicazioni web basata sullo scambio in background di dati tra un web browser e un server.