ODD: Object Design Document

RC - Riconoscimento Carriera

Riferimento	
Versione	0.3
Data	16/12/2019
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Agostino Maria Cassese, Andrea Cella, Gerardo Damiano, Vincenzo De Chiara, Giammarco Fonzo, Lorenzo Maturo, Alessandro Quarto, Gianluca Rossi
Approvato da	Domenico Taffuri, Raffaele De Luca

Revision History

Data	Versione	Descrizione	Autori
13/12/19	0.1	Prima stesura.	AC
15/12/19	0.2	Aggiornamento Class Diagram, Packages	AC
16/12/19	0.3	Inserimento JavaDoc	AC

Sommario

1. Introduzione

Il presente documento illustra l'Object Design per la piattaforma RC. Sono quindi presentate tutte le scelte che precedono la fase di implementazione, come i trade-off su cui ci si è dovuto scontrare e le linee guida a cui i membri del team dovranno attenersi nella fase di implementazione.
Sarà presentata inoltre un'overview dei package e classi del sistema.

1.1 Object Design Trade-off

Nella fase dell'Object Design sorgono diversi compromessi di progettazione ed è necessario stabilire quali punti rispettare e quali rendere opzionali.

Per la piattaforma RC sono stati individuati i seguenti trade-off:

- **Comprensibilità vs Tempo**

  Fondamentale per la progettazione e realizzazione del sistema è la comprensibilità del codice: sia le classi che i metodi dovranno essere semplici, ben documentati e chiari. Inoltre il codice dovrà essere ben indentato e strutturato.

  Di conseguenza, ogni team member dovrà rispettare al massimo le linee guida concordate ed esposte nel prossimo paragrafo.

  Tale comprensibilità, ed il relativo controllo, comporta un aumento del tempo per lo sviluppo e la realizzazione dell'intero sistema, che sarà comunque opportunamente gestito da tutti i membri del team.

- **Interfaccia vs Easy-use**

  L'interfaccia dovrà essere semplice e intuitiva, rispettando i canoni già presenti nel sistema legacy (EV) che estenderemo con la nuova funzionalità identificata con il nome di sistema RC.

  Potrà essere richiesta una penalizzazione in termini di interfaccia per permettere il raggiungimento delle varie funzionalità in pochi passi.

- **Efficienza vs Memoria**

  Essendo che il sistema coinvolge una discreta quantità di dati, si è deciso di utilizzare un compromesso tra efficienza e memoria utilizzata. In caso di query con elevato ritorno di dati, l'attesa per la risposta potrebbe richiedere un tempo leggermente superiore alla media ma questo permette un risparmio in termini di accesso ai dati persistenti e in termini di tabelle ridondanti.

- **Sicurezza vs Prestazioni**

  Nel nostro sistema ogni richiesta del client viene validata attraverso l'uso di sessioni ed un controllo del livello di utenza. Inoltre, si intende utilizzare una libreria per la cifratura della password, in modo tale da poterla inserire nell'URL inviato per e-mail, permettendo quindi l'inserimento dell'utente nel database solo dopo l'effettiva verifica dell'e-mail.
  Questo trade-off resta tuttavia molto bilanciato: anche se le caratteristiche descritte potrebbero far aumentare il tempo di risposta del sistema, il recupero in efficienza potrebbe avvenire in altri casi (accesso al database solo dopo la verifica dell'e-mail).

1.2 Linee guida per l'implementazione

Nell'implementazione del sistema, i membri del team dovranno attenersi alle linee guida di seguito riportate.

1.2.1 Classi e interfacce java

Nella scrittura di codice per le classi Java ci si atterra allo standard "Code Conventions for the Java Programming Language" nella sua interezza, con particolare attenzione a quanto segue:

- Convenzioni sui nomi (cap. 9 dello standard);
- Struttura dei file (cap. 3 dello standard);
- Struttura dei file (cap. 3 dello standard);
- Commenti speciali (sez. 10.5.4 dello standard);
- Utilizzo degli spazi bianchi (cap. 8 dello standard);

Saranno inoltre applicate le seguenti restrizioni e variazioni:

- All'inizio di ogni file dovranno essere presenti alcune informazioni strutturate come segue:

```
/*
    NomeClasse
    Breve descrizione della classe
*/
```

- Tutte le variabili dovrebbero essere private.
- Tutte le variabili che non vengono mai modificate dovrebbero essere dichiarate come costanti (final).
- Si devono usare le tabulazioni per gestire l'indentazione.

1.2.2 Base di dati

Le tabelle della base di dati dovrebbero rispettare la terza forma normale (3NF), in caso non venga rispettato tale vincolo va riportato nella documentazione della base di dati la ragione. Inoltre, anche le scelte sul trattamento dell'integrità referenziale vanno riportate nella documentazione della base di dati.

1.2.3 Java Server Page (JSP)

Le pagine JSP, quando eseguite, devono produrre un documento conforme allo standard HTML versione 5.

Le parti Java delle pagine devono 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;
- È possibile omettere l'uso delle due regole precedenti, se il corpo del codice Java consiste in una singola istruzione:

```
<!Accettabile >
<% for (String par : paragraphs) {%>
<p class='item'><% out.print(par); %></p>
<% } %>
```

```
<!Non accettabile >
<p class='item'><% List<String> paragraphs = getParagraphs();
out.print(paragraphs.get(i++));%></p>
```

1.2.4 Pagine HTML

Le pagine HTML, statiche e dinamiche, devono essere totalmente aderenti allo standard HTML versione 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;

```
<!--Accettabile-- >
<div><span><nl>
    <li>testo</li>
    <li>
        Testo
     </li>
</nl></span></div>
```

```
<!-- Non accettabile -->
<div><span>
<nl>
<li>Uno</li>
<li>
Due
</li>
</nl></span>
</div>
```

- I tag commentati devono seguire le stesse regole che si applicano ai tag normali;

1.2.5 Script JavaScript

Gli script che non svolgono funzioni di rendering della pagina dovrebbero essere collocati in file dedicati.

Il codice Javascript deve seguire le stesse convenzioni per il layout, i nomi ed i commenti del codice Java.

Gli oggetti Javascript devono essere preceduti da un commento in stile Javadoc, che segua il seguente formato:

```
/**
* Descrizione breve
* Eventuale ulteriore descrizione
* Specifica degli argomenti del costruttore (@param)
*
* Metodo nomeMetodo1
* Descrizione breve
* Eventuale ulteriore descrizione
* Specifica degli argomenti (@param)
* Specifica dei risultati (@return)
*
* Metodo nomeMetodo2
```

```
* Descrizione breve
* Eventuale ulteriore descrizione
* Specifica degli argomenti (@param)
* Specifica dei risultati (@return)
*
* ...
*/
function ClasseX(a, b, c) {
```

1.2.6 Fogli di stile CSS

Tutti gli stili non in-line devono essere collocati in fogli di stile separati.

Ogni foglio di stile inizia con un commento simile a quello usato nei file Java.

Le regole CSS vanno formattate 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;

Le proprietà e le regole non chiare saranno precedute da un commento esplicativo.

Le regole possono essere divise in blocchi legati concettualmente, preceduti da commenti in stile Javadoc che ne espongono lo scopo, seguite da 2 righe bianche.

1.2.7 SQL

Le tabelle della base di dati dovrebbero rispettare la terza forma normale (3NF), in caso non venga rispettato tale vincolo va riportato nella documentazione della base di dati la ragione.

I nomi delle tabelle seguiranno le seguenti regole di formattazione:

- Saranno costituiti solo da lettere maiuscole;
- Saranno sostantivi tratti dal dominio del problema che ne esprimono chiaramente il contenuto;

I nomi dei campi seguiranno le seguenti regole di formattazione:

- Saranno costituiti solo da lettere maiuscole;
- In caso di nomi composti da parole multiple, queste saranno separate da underscore (_);
- I nomi saranno sostantivi singolari tratti dal dominio del problema che ne esprimono chiaramente il contenuto;

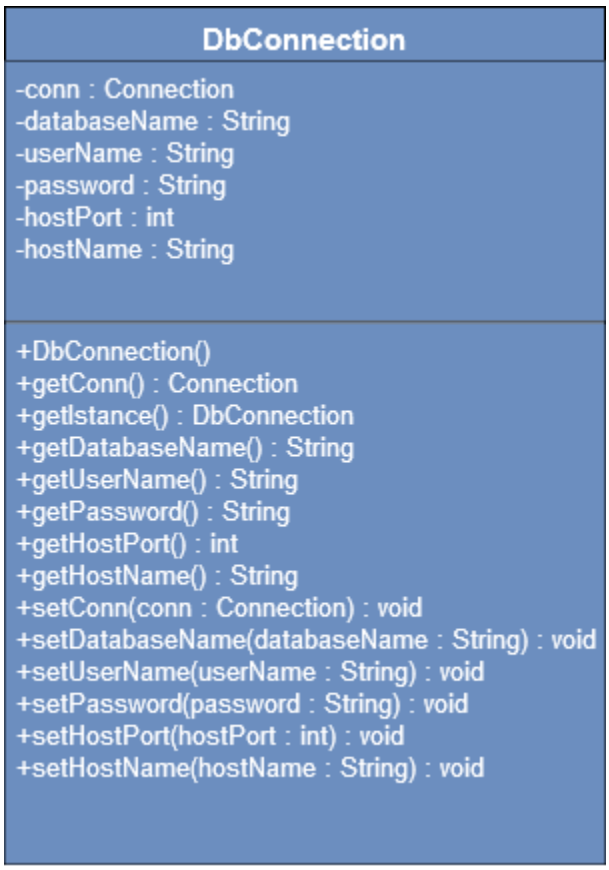
1.3 Design Pattern

1.3.1 MVC

La nostra scelta per il design pattern è stata condizionata dal sistema che andremo ad estendere, English Validation adotta il pattern MVC, per cui anche noi adotteremo lo stesso design pattern sia per non distaccarci troppo dalle scelte progettuali effettuate dal precedente team, sia perché è ideale per un sistema interattivo come quello che andremo ad implementare.
Il design pattern MVC consente di suddividere il sistema in tre blocchi principali: Model, View e Controller. Il Model fornisce i metodi di accesso ai dati persistenti, il View si occupa dell'interazione con l'utente e della presentazione dei dati prelevati dal Model, il Controller riceve i comandi dell'utente attraverso il View e modifica lo stato di quest'ultimo e del Model. Nel nostro sistema le classi sono state divise in package avente nome richiamante il blocco di appartenenza nel design pattern.

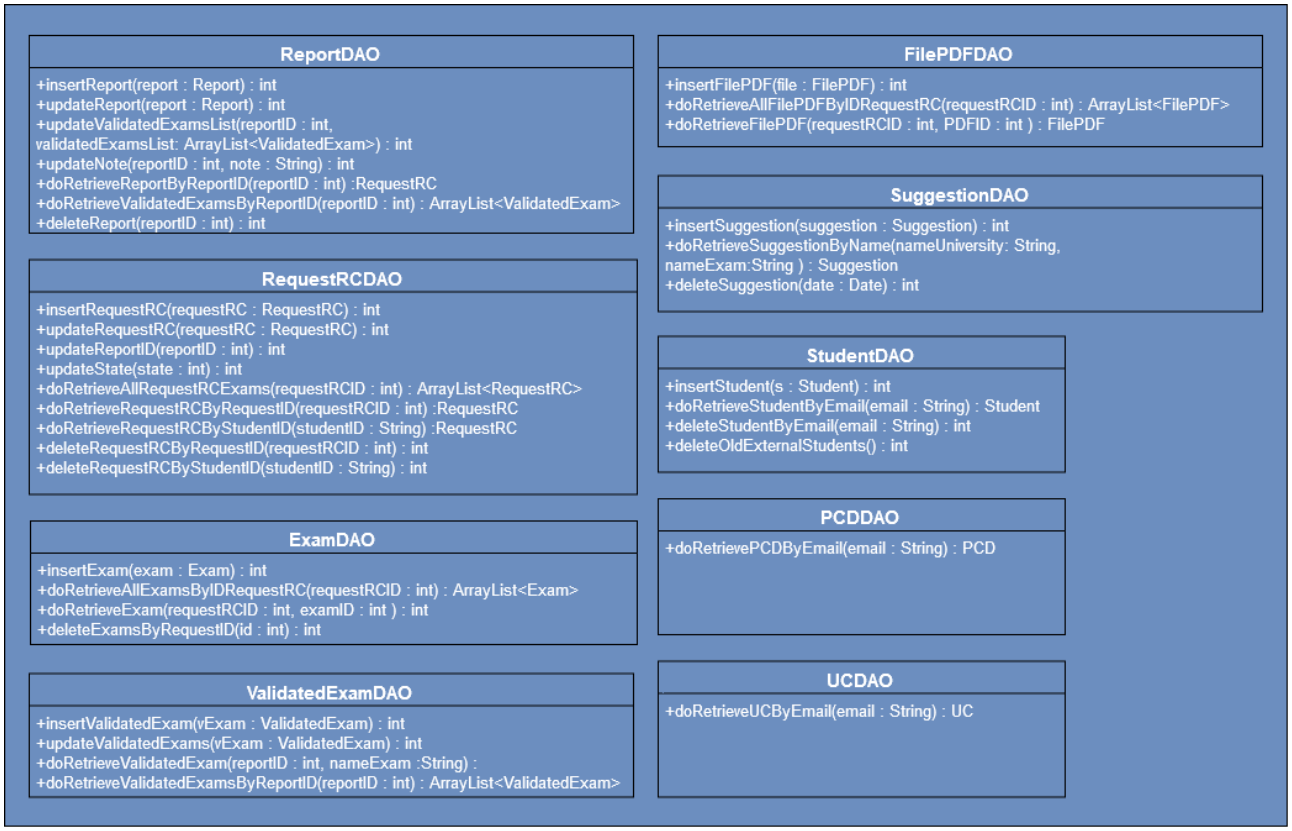
1.3.2 Singleton

L'accesso ai dati persistenti è un'operazione effettuata spesso dal sistema, quindi, per evitarne la criticità, abbiamo progettato una singola classe (DbConnection) che consentisse di effettuare tutte le operazioni con il database. Per evitare la perdita di efficienza dovuta alla creazione di più istanze, si è deciso di applicare a DbConnection il design pattern Singleton.



1.3.3 Data Access Object

L'implementazione del sistema prevede l'utilizzo di un database tramite il DBMS MySQL e le API JDBC per garantire la persistenza dei dati. Per rendere il sistema flessibile ad eventuali futuri cambi di tecnologie è previsto l'utilizzo del design pattern Data Access Object. Il sistema sarà dotato quindi di una classe manager (contraddistinta dal suffisso DAO nel nome) per ogni entità, ognuna di queste classi dovrà offrire tutte le operazioni sull'entità associata tramite un'interfaccia. Le classi controller interagiranno solamente con le interfacce.



1.4 Acronimi

RC: Riconoscimento Carriera

RAD: Requirements Analysis Document

ODD: Object Design Document

SDD: System Design Document

SQL: Structured Query Language

JSP: Java Server Pages

HTML: HypertText Markup Language

CSS: Cascade Style Sheets

3NF: Terza forma normale

COTS: Components Off-The-Shelf

API: Application Programming Interface

1.5 Riferimenti

Il presente ODD fa riferimento alle ultime versioni dei documenti rilasciati:

- RC_RAD_v_1.9
- RC_SDD_v_0.1

Si fa riferimento, inoltre, alle presentazioni della Prof. Ferrucci ed ai seguenti libri di testo:

- B.Bruegge, A.H. Dutoit, Object Oriented Software Engineering – Using UML, Patterns and Java, Prentice Hall.
- C. Ghezzi, D. Mandrioli, M. Jazayeri, Ingegneria del Software – Fondamenti e Principi, Prentice Hall.
- Sommerville, Software Engineering, Addison Wesley.
- R.S. Pressman Principi di Ingegneria del Software, Mc Graw Hill.
- Jim Arlow, Ila Neustadt, UML e Unified Process, McGraw-Hill.

1.6 Organizzazione del contenuto

In questo primo capitolo si è fatta un'introduzione al documento, trattando i vari trade-off e le linee guida per l'implementazione. Il capitolo 2 mostrerà la suddivisione in classi e package, e nel capitolo 3 saranno presentate le tecniche utilizzate per garantire il riuso, come i COTS ed i Design Pattern.

Il documento si conclude con un glossario.

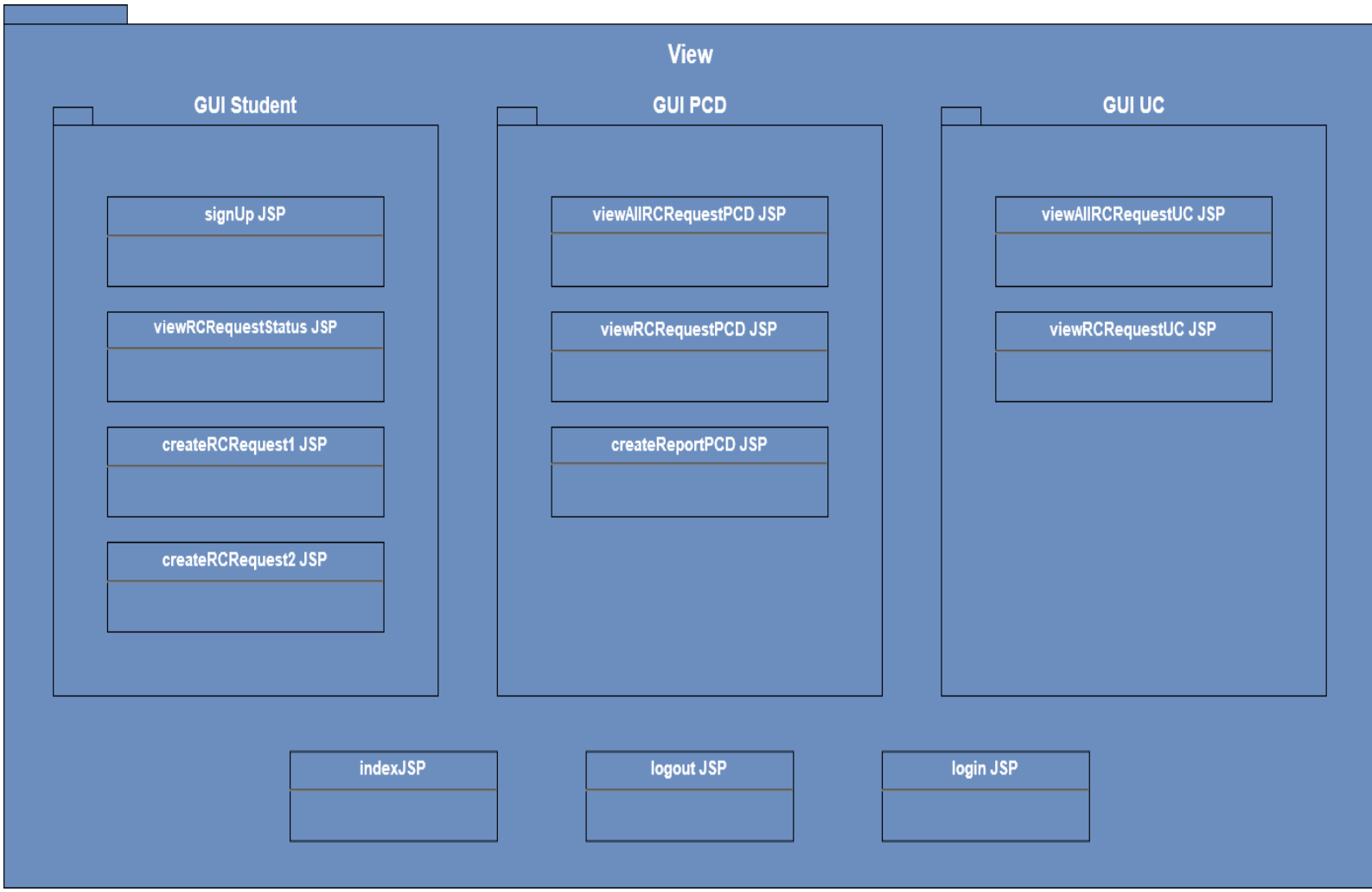
2. Packages

2.1 View

Il package View contiene tre package: GUI Student, GUI PCD, GUI UC e oltre a questi package presenta delle classi che sono Login JSP, Logout JSP, Index JSP. Il package Student viene implementato con le classi: SignUp JSP, ViewRCRequestStatus JSP, CreateRCRequest1 JSP, CreateRCRequest2 JSP. SignUp JSP viene utilizzata per fornire la funzione di registrazione al sistema. ViewRequestStatus JSP viene utilizzata per mostrare allo studente lo status di avanzamento della sua richiesta, questa pagina viene mostrata solo se è già stata inoltrata una richiesta da parte dello studente che accede al sistema. CreateRCRequest1 e CreateRCRequest2 vengono utilizzate per fornire la funzione di invio di una nuova richiesta (inserendo tutti i relativi dati richiesti senza errori), la pagina CreateRCRequest1 viene mostrata allo studente dopo la sua autenticazione inoltre, lo studente in questione non deve essere inscritto presso l'università degli studi di Salerno (studente esterno) e non deve essere presente una richiesta precedente. Per quanto riguarda gli studenti dell'università degli studi di Salerno (studenti interni) la pagina CreateRCRequest1 viene mostrata dopo l'autenticazione e la scelta, tramite il pulsante presente nella navbar, della funzione di riconoscimento carriera e se non è già presente una richiesta precedente. La pagina CreateRCRequest2 viene mostrata agli studenti (indipendentemente dal fatto che siano interni/esterni) dopo che la prima parte di dati della richiesta saranno inseriti correttamente nella pagina CreateRCRequest1 JSP. Il package PCD viene implementato con le classi: ViewAllRCRequestPCD JSP, ViewRCRequestPCD JSP e CreateReportPCD JSP. La pagina ViewAllRCRequestPCD permette la visualizzazione della lista delle richieste pervenute (richieste che hanno superato la validazione dell'UC), la pagina viene mostrata dopo che il PCD si è autenticato e ha scelto dalla navbar la sezione di riconoscimento carriera. La pagina ViewRCRequestPCD permette la visualizzazione di una singola richiesta in una schermata divisa in due sezioni, a destra viene visualizzato il file PDF caricato dallo studente, contenente la carriera pregressa certificata dall'università, a sinistra viene visualizzata la lista degli esami inseriti manualmente dallo studente nella fase di creazione ed inoltro della richiesta. La suddetta pagina inoltre permette di visualizzare eventuali suggerimenti per un singolo esame, inviare una mail precompilata (contenente un riferimento al programma dell'esame) ad un docente e infine la visualizzazione del programma relativo al singolo esame. La pagina CreateReportPCD permette al PCD di inserire i CFU che vengono convalidati per i singoli esami, di specificare con quali esami (esami relativi al curriculum del dipartimento di informatica dell'università degli studi di Salerno) vengono convalidati quelli presentati dallo studente ed infine permette la creazione e l'invio di un report allo studente con eventuali note sulle modalità di convalida utilizzate. Il

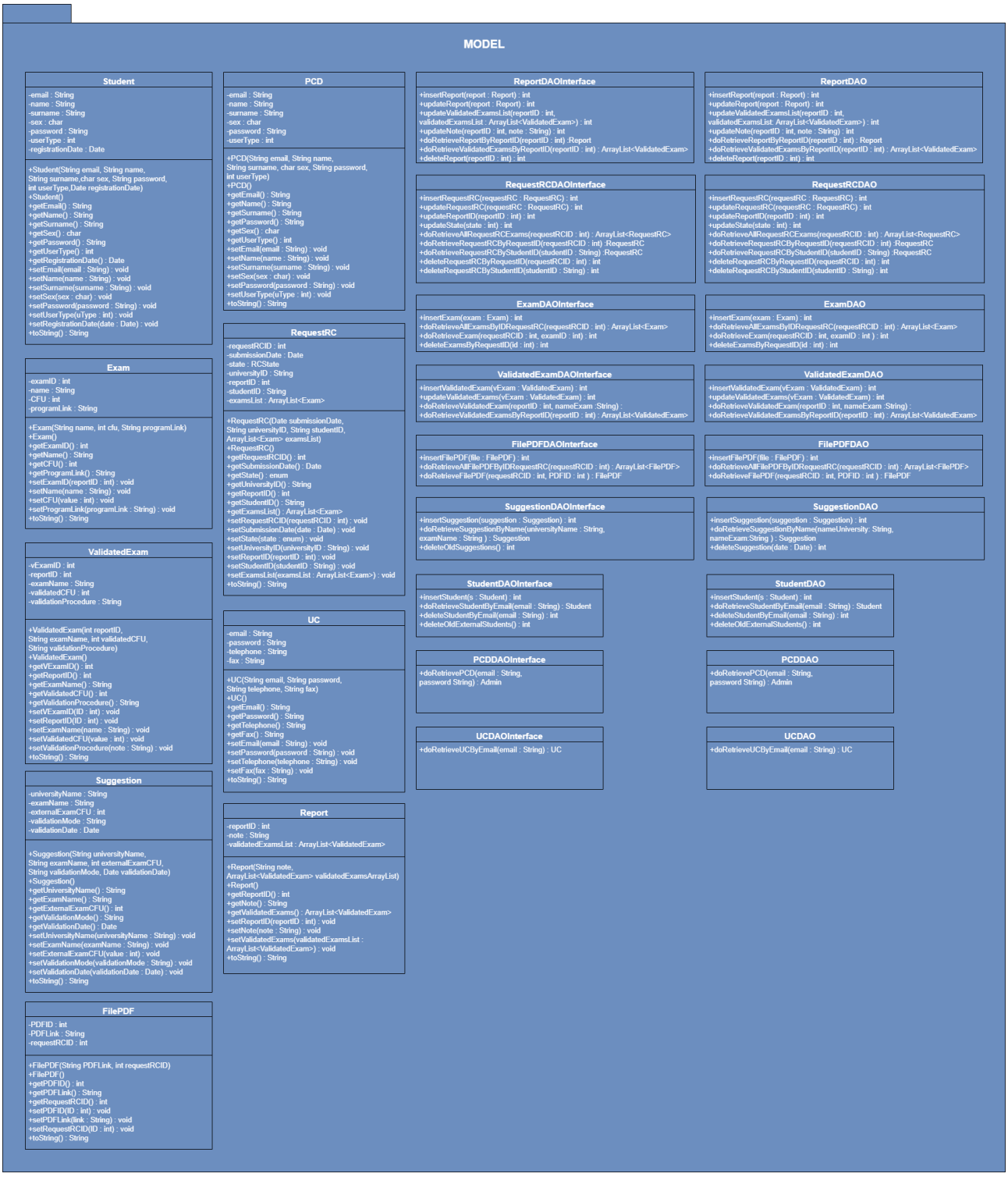
package UC viene implementato con le classi: ViewAllRCRequestUC JSP e ValidateRCRequestUC. La pagina ViewAllRCRequestUC permette la visualizzazione di tutte le richieste pervenute. La pagina

ValidateRCRequestUC permette di visualizzare una singola richiesta con annessi i file PDF allegati (dei quali si può effettuare un download) e permette la validazione della richiesta che in caso di validazione positiva viene passata al PCD. La pagina Login JSP consente l'autenticazione e l'accesso al sistema tramite le proprie credenziali, la pagina Logout JSP consente di uscire dal sistema Infine, la pagina Index JSP è la prima pagina che viene mostrate all'avvio del sistema. Le JSP che sono al di fuori dai sotto-package: GUI Student, GUI PCD, GUI UC sono da considerare generali e quindi fruibili da tutti e tre i tipi di user. Il package View comunica, inviando i comandi degli utenti, con il package Controller.



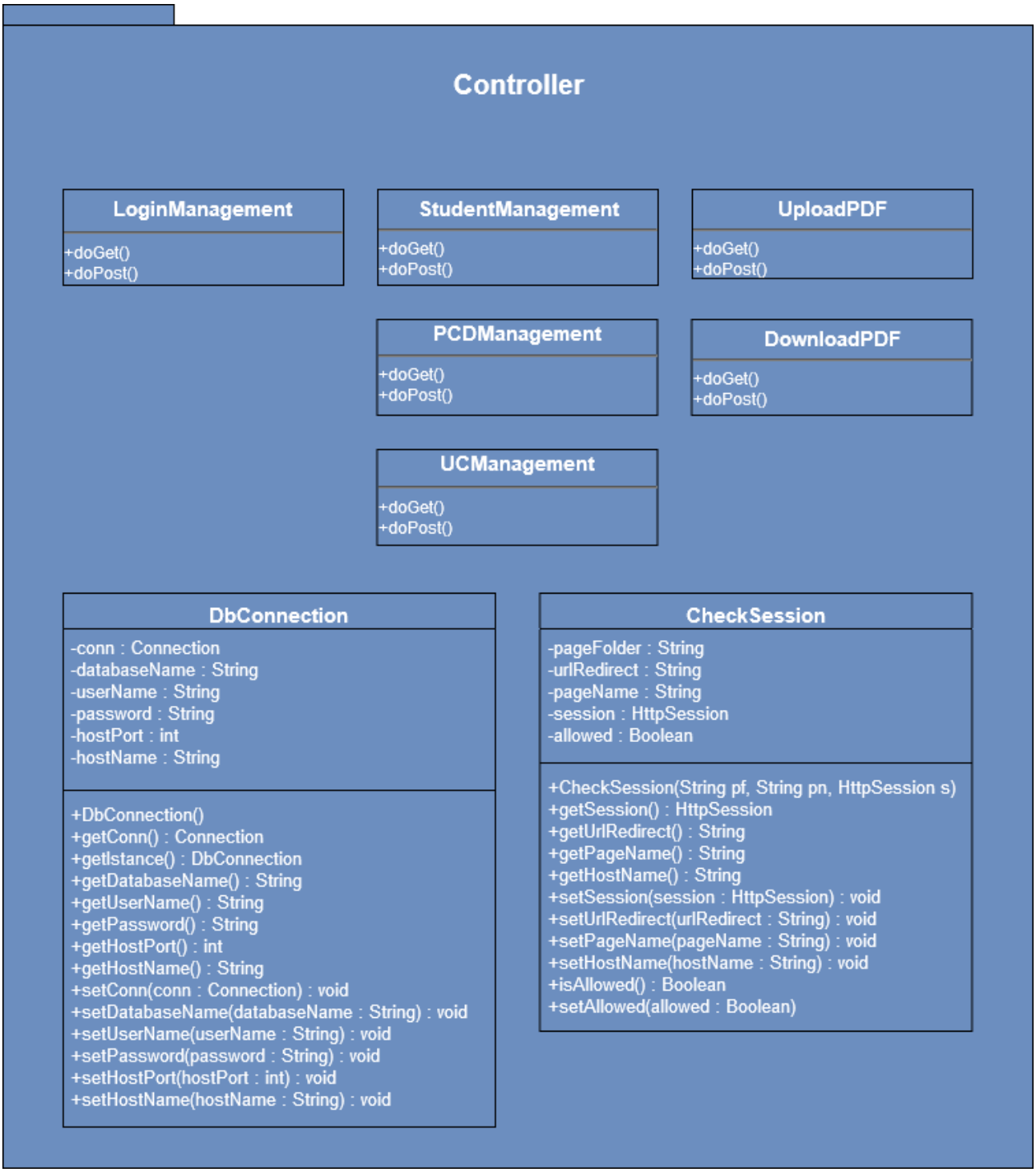
2.2 Model

Il package Model contiene tutte le classi per la gestione dei dati persistenti, per le interazioni con il database e le classi che definiscono gli oggetti che sui quali lavoriamo. Il package è implementato tramite le classi: Student, PCD, UC, Exam, RequestRC, Report, FilePDF, ValidateExam, Suggestion che definiscono gli oggetti sui quali lavoriamo e dalle classi: StudenteDAO, PCDDAO, UCDAO, ReportDAO, FilePDFDAO, ValidateExamDAO, SuggestionDAO, ExamDAO, StudenteDAOInterface, PCDDAOInterface, UCDAOInterface, ReportDAOInterface, FilePDFDAOInteface, ValidateExamDAOInterface, SuggestionDAOInterface, ExamDAOInterface che vengono utilizzate per le interazioni con il database.


MODEL
Student
-email : String
-name : String
-surname : String
-sex : char
-password : String
-userType : int
-registrationDate : Date
+Student(String email, String name, String surname,char sex, String password, int userType,Date registrationDate)
+Student()
+getEmail() : String
+getName() : String
+getSurname() : String
+getSex() : char
+getPassword() : String
+getUserType() : int
+getRegistrationDate() : Date
+setEmail(email : String) : void
+setName(name : String) : void
+setSurname(surname : String) : void
+setSex(sex : char) : void
+setPassword(password : String) : void
+setUserType(uType : int) : void
+setRegistrationDate(date : Date) : void
+toString() : String
Exam
-examID : int
-name : String
-CFU : int
-programLink : String
+Exam(String name, int cfu, String programLink)
+Exam()
+getExamID() : int
+getName() : String
+getCFU() : int
+getProgramLink() : String
+setExamID(reportID : int) : void
+setName(name : String) : void
+setCFU(value : int) : void
+setProgramLink(programLink : String) : void
+toString() : String
ValidatedExam
-vExamID : int
-reportID : int
-examName : String
-validatedCFU : int
-validationProcedure : String
+ValidatedExam(int reportID, String examName, int validatedCFU, String validationProcedure)
+ValidatedExam()
+getVExamID() : int
+getReportID() : int
+getExamName() : String
+getValidatedCFU() : int
+getValidationProcedure() : String
+setVExamID(ID : int) : void
+setReportID(ID : int) : void
+setExamName(name : String) : void
+setValidatedCFU(value : int) : void
+setValidationProcedure(note : String) : void
+toString() : String
Suggestion
-universityName : String
-examName : String
-externalExamCFU : int
-validationMode : String
-validationDate : Date
+Suggestion(String universityName, String examName, int externalExamCFU, String validationMode, Date validationDate)
+Suggestion()
+getUniversityName() : String
+getExamName() : String
+getExternalExamCFU() : int
+getValidationMode() : String
+getValidationDate() : Date
+setUniversityName(universityName : String) : void
+setExamName(examName : String) : void
+setExternalExamCFU(value : int) : void
+setValidationMode(validationMode : String) : void
+setValidationDate(validationDate : Date) : void
+toString() : String
FilePDF
-PDFID : int
-PDFLink : String
-requestRCID : int
+FilePDF(String PDFLink, int requestRCID)
+FilePDF()
+getPDFID() : int
+getPDFLink() : String
+getRequestRCID() : int
+setPDFID(ID : int) : void
+setPDFLink(link : String) : void
+setRequestRCID(ID : int) : void
+toString() : String
PCD
-email : String
-name : String
-surname : String
-sex : char
-password : String
-userType : int
+PCD(String email, String name, String surname, char sex, String password, int userType)
+PCD()
+getEmail() : String
+getName() : String
+getSurname() : String
+getPassword() : String
+getSex() : char
+getUserType() : int
+setEmail(email : String) : void
+setName(name : String) : void
+setSurname(surname : String) : void
+setSex(sex : char) : void
+setPassword(password : String) : void
+setUserType(uType : int) : void
+toString() : String
RequestRC
-requestRCID : int
-submissionDate : Date
-state : RCState
-universityID : String
-reportID : int
-studentID : String
-examsList : ArrayList<Exam>
+RequestRC(Date submissionDate, String universityID, String studentID, ArrayList<Exam> examsList)
+RequestRC()
+getRequestRCID() : int
+getSubmissionDate() : Date
+getState() : enum
+getUniversityID() : String
+getReportID() : int
+getStudentID() : String
+getExamsList() : ArrayList<Exam>
+setRequestRCID(requestRCID : int) : void
+setSubmissionDate(date : Date) : void
+setState(state : enum) : void
+setUniversityID(universityID : String) : void
+setReportID(reportID : int) : void
+setStudentID(studentID : String) : void
+setExamsList(examsList : ArrayList<Exam>) : void
+toString() : String
UC
-email : String
-password : String
-telephone : String
-fax : String
+UC(String email, String password, String telephone, String fax)
+UC()
+getEmail() : String
+getPassword() : String
+getTelephone() : String
+getFax() : String
+setEmail(email : String) : void
+setPassword(password : String) : void
+setTelephone(telephone : String) : void
+setFax(fax : String) : void
+toString() : String
Report
-reportID : int
-note : String
-validatedExamsList : ArrayList<ValidatedExam>
+Report(String note, ArrayList<ValidatedExam> validatedExamsArrayList)
+Report()
+getReportID() : int
+getNote() : String
+getValidatedExams() : ArrayList<ValidatedExam>
+setReportID(reportID : int) : void
+setNote(note : String) : void
+setValidatedExams(validatedExamsList : ArrayList<ValidatedExam>) : void
+toString() : String
ReportDAOInterface
+insertReport(report : Report) : int
+updateReport(report : Report) : int
+updateValidatedExamsList(reportID : int, validatedExamsList : ArrayList<ValidatedExam>) : int
+updateNote(reportID : int, note : String) : int
+doRetrieveReportByReportID(reportID : int) :Report
+doRetrieveValidatedExamsByReportID(reportID : int) : ArrayList<ValidatedExam>
+deleteReport(reportID : int) : int
ReportDAO
+insertReport(report : Report) : int
+updateReport(report : Report) : int
+updateValidatedExamsList(reportID : int, validatedExamsList :ArrayList<ValidatedExam>) : int
+updateNote(reportID : int, note : String) : int
+doRetrieveReportByReportID(reportID : int) : Report
+doRetrieveValidatedExamsByReportID(reportID : int) : ArrayList<ValidatedExam>
+deleteReport(reportID : int) : int
RequestRCDAOInterface
+insertRequestRC(requestRC : RequestRC) : int
+updateRequestRC(requestRC : RequestRC) : int
+updateReportID(reportID : int) : int
+updateState(state : int) : int
+doRetrieveAllRequestRCExams(requestRCID : int) : ArrayList<RequestRC>
+doRetrieveRequestRCByRequestID(requestRCID : int) :RequestRC
+doRetrieveRequestRCByStudentID(studentID : String) :RequestRC
+deleteRequestRCByRequestID(requestRCID : int) : int
+deleteRequestRCByStudentID(studentID : String) : int
RequestRCDAO
+insertRequestRC(requestRC : RequestRC) : int
+updateRequestRC(requestRC : RequestRC) : int
+updateReportID(reportID : int) : int
+updateState(state : int) : int
+doRetrieveAllRequestRCExams(requestRCID : int) : ArrayList<RequestRC>
+doRetrieveRequestRCByRequestID(requestRCID : int) :RequestRC
+doRetrieveRequestRCByStudentID(studentID : String) :RequestRC
+deleteRequestRCByRequestID(requestRCID : int) : int
+deleteRequestRCByStudentID(studentID : String) : int
ExamDAOInterface
+insertExam(exam : Exam) : int
+doRetrieveAllExamsByIDRequestRC(requestRCID : int) : ArrayList<Exam>
+doRetrieveExam(requestRCID : int, examID : int) : int
+deleteExamsByRequestID(id : int) : int
ExamDAO
+insertExam(exam : Exam) : int
+doRetrieveAllExamsByIDRequestRC(requestRCID : int) : ArrayList<Exam>
+doRetrieveExam(requestRCID : int, examID : int) : int
+deleteExamsByRequestID(id : int) : int
ValidatedExamDAOInterface
+insertValidatedExam(vExam : ValidatedExam) : int
+updateValidatedExams(vExam : ValidatedExam) : int
+doRetrieveValidatedExam(reportID : int, nameExam :String) :
+doRetrieveValidatedExamsByReportID(reportID : int) : ArrayList<ValidatedExam>
ValidatedExamDAO
+insertValidatedExam(vExam : ValidatedExam) : int
+updateValidatedExams(vExam : ValidatedExam) : int
+doRetrieveValidatedExam(reportID : int, nameExam :String) :
+doRetrieveValidatedExamsByReportID(reportID : int) : ArrayList<ValidatedExam>
FilePDFDAOInterface
+insertFilePDF(file : FilePDF) : int
+doRetrieveAllFilePDFByIDRequestRC(requestRCID : int) : ArrayList<FilePDF>
+doRetrieveFilePDF(requestRCID : int, PDFID : int) : FilePDF
FilePDFDAO
+insertFilePDF(file : FilePDF) : int
+doRetrieveAllFilePDFByIDRequestRC(requestRCID : int) : ArrayList<FilePDF>
+doRetrieveFilePDF(requestRCID : int, PDFID : int) : FilePDF
SuggestionDAOInterface
+insertSuggestion(suggestion : Suggestion) : int
+doRetrieveSuggestionByName(universityName : String, examName : String) : Suggestion
+deleteOldSuggestions() : int
SuggestionDAO
+insertSuggestion(suggestion : Suggestion) : int
+doRetrieveSuggestionByName(nameUniversity: String, nameExam:String) : Suggestion
+deleteSuggestion(date : Date) : int
StudentDAOInterface
+insertStudent(s : Student) : int
+doRetrieveStudentByEmail(email : String) : Student
+deleteStudentByEmail(email : String) : int
+deleteOldExternalStudents() : int
StudentDAO
+insertStudent(s : Student) : int
+doRetrieveStudentByEmail(email : String) : Student
+deleteStudentByEmail(email : String) : int
+deleteOldExternalStudents() : int
PCDDAOInterface
+doRetrievePCD(email : String, password String) : Admin
PCDDAO
+doRetrievePCD(email : String, password String) : Admin
UCDAOInterface
+doRetrieveUCByEmail(email : String) : UC
UCDAO
+doRetrieveUCByEmail(email : String) : UC

2.3 Controller

Il package Controller riceve, tramite il package View, i comandi dell'utente. Il package Controller è implementato tramite le classi: LoginLogoutManagement, DbConnection, UploadPDF, DownloadPDF, CheckSession, StudentManagement, PCDManagement, UCManagement. StudentManagement si occupa di gestire i comandi lanciati dagli studenti (comunica con UserInterface, StudentDAO, UploadPDF, LoginLogoutManagement, RequestDAO). PCDManagement gestisce i comandi lanciati dal PCD (comunica con UserInterface, PDCDAO, RequestDAO, LoginLogoutManagement). UCManagement gestisce i comandi lanciati dall'UC (comunica con UCDAO, RequestDAO, DownloadPDF, LoginLogoutManagement). DownloadPDF permette lo scaricamento degli allegati. UploadPDF permette il caricamento degli allegati. DbConnection si occupa delle interazioni con il database. CheckSession si occupa di indirizzare l'utente alla pagina a lui destinata. LoginLogoutManagement gestisce la procedura di autenticazione degli utenti e la procedura di uscita dal sistema.


Controller
LoginManagement
+doGet()
+doPost()
StudentManagement
+doGet()
+doPost()
UploadPDF
+doGet()
+doPost()
PCDManagement
+doGet()
+doPost()
DownloadPDF
+doGet()
+doPost()
UCManagement
+doGet()
+doPost()
DbConnection
-conn : Connection
-databaseName : String
-userName : String
-password : String
-hostPort : int
-hostName : String
+DbConnection()
+getConn() : Connection
+getIstance() : DbConnection
+getDatabaseName() : String
+getUserName() : String
+getPassword() : String
+getHostPort() : int
+getHostName() : String
+setConn(conn : Connection) : void
+setDatabaseName(databaseName : String) : void
+setUserName(userName : String) : void
+setPassword(password : String) : void
+setHostPort(hostPort : int) : void
+setHostName(hostName : String) : void
CheckSession
-pageFolder : String
-urlRedirect : String
-pageName : String
-session : HttpSession
-allowed : Boolean
+CheckSession(String pf, String pn, HttpSession s)
+getSession() : HttpSession
+getUrlRedirect() : String
+getPageName() : String
+getHostName() : String
+setSession(session : HttpSession) : void
+setUrlRedirect(urlRedirect : String) : void
+setPageName(pageName : String) : void
+setHostName(hostName : String) : void
+isAllowed() : Boolean
+setAllowed(allowed : Boolean)

3. Class Interface

Nome Classe	Index JSP
Descrizione	Questa classe accoglie l'utente
Pre-condizione	NA
Post-condizione	NA
Invariante	NA

Nome Classe	Login JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa al processo di login.
Pre-condizione	context Login JSP: validate(email, Password)0; pre: email!=null && password!=null email.equals(DB.email) && password.equals(DB.password)
Post-condizione	NA
Invariante	NA

Nome Classe	Logout JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa al processo di logout.
Pre-condizione	context Login JSP:Logout(); pre: isLogged();
Post-condizione	NA
Invariante	NA

Nome Classe	SignUpJSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa al processo registrazione.
Pre-condizione	context Login JSP: signUp(String e-mail, String name, String surname, String password); pre: validate(String e-mail, String name,

	String surname, String password)
Post-condizione	NA
Invariante	NA

Nome Classe	ViewRCRequestStatus JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa alla visualizzazione dello stato di avanzamento della richiesta inoltrata.
Pre-condizione	context ViewRCRequestStatus JSP: request.getStatus();
Post-condizione	NA
Invariante	NA

Nome Classe	createRCRequest1 JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa alla compilazione della prima parte della richiesta.

Pre-condizione	context createRCRequest1: validate(String nameUni);
	isPDF(PDF1);
	isPDF(PDF2)

Post-condizione	context createRCRequest1: activeButton("buttonNext")
Invariante	NA

Nome Classe	createRCRequest2 JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa alla compilazione della seconda parte della richiesta.
Pre-condizione	context createRCRequest2: validateExam(String name, int cfu, String reference)
Post-condizione	context createRCRequest2: activeButton("buttonSubmit")
Invariante	NA

Nome Classe	viewAllRCRequestPCD JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa alla visualizzazione delle richieste.
Pre-condizione	Il PCD è loggato e clicca sul tasto "Richieste RC"
Post-condizione	NA
Invariante	NA

Nome Classe	viewRCRequestPCD JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa alla visualizzazione della singola richiesta.
Pre-condizione	context viewAllRCRequestPCD JSP: onClick("viewRequest").
Post-condizione	context viewRCRequestPCD JSP: showExamList(List list);
Invariante	NA

Nome Classe	createReportPCD JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa alla compilazione del report relativo alla richiesta ed il suo invio allo studente.
Pre-condizione	contextt createReportPCD JSP: showExamList(List list); validate(int CFUApproved, String modeConvalidation,String note);
Post-condizione	NA
Invariante	NA

Nome Classe	viewAllRCRequestUC JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa alla visualizzazione delle richieste pervenute all'UC.
Pre-condizione	L'UC è loggato.
Post-condizione	NA
Invariante	NA

Nome Classe	validateRCRequestUC JSP
Descrizione	Questa classe rappresenta il gestore della funzionalità relativa alla visualizzazione della singola richiesta.
Pre-condizione	viewAllRCRequestUC JSP: onClick("CheckRequest")
Post-condizione	NA
Invariante	NA

Nome Classe	RequestRC
Descrizione	Mantiene i dati della richiesta di riconoscimento della carriera pregressa.
Pre-condizione	*NA*
Post-condizione	*NA*
Invariante	*NA*

Nome Classe	RequestRCDAO
Descrizione	Interagisce con il database al fine di gestire i dati della richiesta di riconoscimento della carriera pregressa.
Pre-condizione	*NA*
Post-condizione	*NA*
Invariante	I dati persistenti relativi alla data di creazione della richiesta, all'università di provenienza, allo studente che l'ha inoltrata, all'ID della richiesta e alla lista degli esami contenuti all'interno della richiesta, presenti nel database non possono essere modificati.

Nome Classe	RequestRCDAOInterface
Descrizione	Offre l'interfaccia al RequestRCDAO.
Pre-condizione	*NA*
Post-condizione	*NA*
Invariante	*NA*

Nome Classe	Report
Descrizione	Mantiene i dati del report relativo ad una richiesta di riconoscimento della carriera pregressa.
Pre-condizione	*NA*
Post-condizione	*NA*
Invariante	Il reportID non può essere modificato.

Nome Classe	ReportDAO
Descrizione	Interagisce con il database al fine di gestire i dati del report relativo ad una richiesta di riconoscimento della carriera pregressa.
Pre-condizione	*NA*
Post-condizione	*NA*
Invariante	I dati persistenti relativi alla data di creazione della richiesta, all'università di provenienza, allo studente che l'ha inoltrata, all'ID della richiesta e alla lista degli esami contenuti all'interno della richiesta, presenti nel database non possono essere modificati.

Nome Classe	ReportDAOInterface
Descrizione	Offre l'interfaccia al ReportDAO.
Pre-condizione	*NA*
Post-condizione	*NA*
Invariante	*NA*

Nome Classe	LoginLogoutManagement
Descrizione	Gestisce il login e il logout degli utenti dal sistema.
Pre-condizione	*NA*
Post-condizione	*NA*
Invariante	*NA*

Nome Classe	StudentManagement
Descrizione	Gestisce le operazioni che lo studente può richiedere al sistema.
Pre-condizione	*NA*
Post-condizione	*NA*
Invariante	*NA*

Nome Classe	PCDManagement
Descrizione	Gestisce le operazioni che il PCD può richiedere al sistema.
Pre-condizione	*NA*
Post-condizione	*NA*
Invariante	*NA*

Nome Classe	UCManagement
Descrizione	Gestisce le operazioni che l'addetto UC può richiedere al sistema
Pre-condizione	*NA*
Post-condizione	*NA*
Invariante	*NA*

Nome Classe	UploadPDF
Descrizione	Permette il caricamento dei file PDF da parte dello studente durante l'invio della richiesta di riconoscimento della carriera pregressa.
Pre-condizione	Il file da caricare e salvare nel database deve essere in formato PDF.
Post-condizione	*NA*
Invariante	*NA*

Nome Classe	DownloadPDF
Descrizione	Permette il download dei file PDF da parte dell'UC durante l'invio della richiesta di riconoscimento della carriera pregressa.
Pre-condizione	Il file da scaricare deve esistere nel sistema.
Post-condizione	*NA*
Invariante	*NA*

Nome Classe	DBConnection
Descrizione	Permette al sistema di stabilire una connessione con il database.
Pre-condizione	Il server del database deve essere avviato e accessibile.
Post-condizione	*NA*
Invariante	*NA*

Nome Classe	CheckSession
Descrizione	Gestisce la sessione.
Pre-condizione	*NA*
Post-condizione	*NA*
Invariante	*NA*

Per ulteriori dettagli si faccia riferimento al JavaDoc, che contiene tutti i dettagli sull'implementazione delle classi e dei metodi.

4. Class Diagram

All'interno del nostro sistema le classi che si occupano di gestire le funzionalità relative al PCD hanno come denominazione "PCD", invece, all'interno del codice il nome è stato cambiato in "Admin" per rispettare gli standard di nomenclatura fornitesi dal sistema legacy.

5. Glossario

RC: Riconoscimento Carriera

RAD: Requirements Analysis Document

ODD: Object Design Document

SDD: System Design Document

SQL: Structured Query Language

JSP: Java Server Pages

HTML: HypertText Markup Language

CSS: Cascade Style Sheets

3NF: Terza forma normale

COTS: Components Off-The-Shelf

API: Application Programming Interface