UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Corso di Ingegneria del Software

OBJECT DESIGN DOCUMENT

NOMI PARTECIPANTI	MATRICOLE PARTECIPANTI
DE FILIPPO GAETANO	**0512105928**
D'AVINO GIUSEPPE	**0512105976**
MONTEFUSCO ANTONIO RENATO	**0512105806**
CASULLO PAMELA MAURA	**0512105808**

1- Introduzione

1.1 Object design trade offs

Comprensibilità vs Tempo:
Il codice del sistema deve essere comprensibile, in modo da facilitare la fase di testing ed eventuali future modifiche da apportare.

Prestazioni vs Costi:
Dal momento che il budget allocato è spendibile principalmente in risorse umane e non consente l'acquisto di tecnologie proprietarie specifiche, verranno utilizzati template open source e componenti hardware di nostra proprietà

Interfaccia vs Usabilità:
Verrà realizzata un'interfaccia chiara e user friendly, usando form e pulsanti predefiniti che hanno lo scopo di rendere semplice l'utilizzo del sistema da parte dell'utente finale.

Sicurezza vs Efficienza:
La sicurezza per via di tempi di sviluppo molto limitati, ci limiteremo ad implementare sistemi di sicurezza basati su filtri, e-mail e password criptate in SHA-56.

1.2 Componenti off-the-shelf

Per l'implementazione del sistema utilizzeremo componenti software già disponibili e utilizzati per facilitare la creazione del software.

- **Bootstrap**, framework open source utilizzato per sviluppare progetti web. Verrà utilizzato insieme ad HTML, CSS e JavaScript.
- **jQuery,** libreria JavaScript che facilita la scrittura di scrip rendendo semplice la selezione e la manipolazione di elementi del DOM in pagine HTML.
- **AJAX**, tecnica di sviluppo software per la realizzazione di applicazioni web interattive che utilizzano scambi tra web browser e server. Consente l'aggiornamento dinamico di una pagina web senza caricamento esplicito da parte dell'utente.Verrà utilizzato insieme all'estensione **JSON.**
- **JSON,** formato di dati adatto allo scambio di informazioni in applicativi client/server. Usato in **AJAX** tramite l'API **XHRHttp**
- **Selenium,** una suite di tool utilizzati per automatizzare i test di sistema eseguendoli sul web browser.
- **JUnit,** un framework di programmazione Java che viene utilizzato per implementare i test di unità
- **JavaMail API,** API per inviare email da codice java
- **Azure,** piattaforma microsoft che offre servizi di cloud dove verrà hostato il database

1.3 Linee guida

Gli sviluppatori dovranno seguire precise linee guida per la stesura del codice:

Package

Il progetto verrà sviluppato con L'IDE Eclipse, sarà strutturato come segue:

- Il progetto avrà 3 package; **model, control e test**, i quali contengono i corrispettivi subpackage con rispettive classi.

Naming Convention

Per la documentazione delle interfacce bisognerà utilizzare nomi:

- Descrittivi;
- Pronunciabili;
- Di lunghezza medio-corta;
- Caratteri consentiti (a-z,A-Z,0-9)

Variabili

I nomi delle variabili dovranno seguire l'annotazione "**Camel Notation**", quindi prima lettera minuscola e le successive parole con l'iniziale maiuscola. Tipo **camelCase**.

In ogni riga dovrà esserci un'unica variabile dichiarata, eventualmente allineata con quelle del blocco dichiarativo.

Costanti

I nomi delle costanti dovranno seguire l'annotazione " CONSTANT_CASE", tutte le lettere maiuscole, le parole saranno separate da underscore "_".

Le costanti di classe dovranno essere dichiarate all'inizio della classe.

Metodi

I nomi dei metodi dovranno seguire la notazione "**Camel Notation**", quindi prima lettera minuscola e le successive parole con l'iniziale maiuscola. Tipo **camelCase**.

Il nome del metodo sarà costituito da un verbo che ne identifica l'azione seguito da un sostantivo, eventualmente aggettivato.

I metodi dovranno essere descritti tramite Javadoc.

Classi Java

I nomi delle classi seguiranno le convenzioni della **OOP** detta anche "**Pascal Case"**(camelCase con prima lettera maiuscola), quindi prima lettera maiuscola e le successive parole con l'iniziale maiuscola.

I nomi delle classi dovranno essere semplici, descrittivi e inerenti al dominio applicativo. Vietato usare underscore.

I nomi dei metodi accessori e modificatori seguiranno i pattern standard della **OOP**, rispettivamente saranno, **getNomeVariabile** e **setNomeVariabile**.

Le classi saranno strutturate nel seguente ordine:

- Dichiarazione della classe pubblica;
- Dichiarazioni di costanti;
- Dichiarazioni di variabili di classe;
- Dichiarazioni di variabili di istanza;
- Costruttore;
- Metodi pubblici;
- Metodi di servizio privati;

```java
/**
 * @author Antonio
 * Questo è il Javadoc della classe "ClasseProva"
 */
public class ClasseProva {

    private final static String COSTANTE="Questa è la costante di prova";
    private static int  variabileStatic=0;//Variabile di classe
    private String variabile;   //Variabile di istanza
    //Costruttore vuoto
    public ClasseProva() {
        this.variabile="";
    }
    //Costruttore con variabili di classe
    public ClasseProva(String variabile) {
        this.variabile = variabile;
    }

    //Metodi pubblici
    public String getVariabile() {
        return variabile;
    }
    public void setVariabile(String variabile) {
        this.variabile = variabile;
    }
    public static int getVariabileStatic() {
        return variabileStatic;
    }
    /**
     * Questo è il Javadoc del metodo "somma()"
     */
    public void somma() {
        Scanner scan=new Scanner(System.in);
        aggiungi(scan.nextInt());
    }

    //Metodi privati
    private void aggiungi(int x) {
        this.variabileStatic+=x;
    }
}
```

Pagine JSP

I nomi delle pagine JSP dovranno seguire la notazione “**camelCase”** descritta per i metodi delle classi Java.

Le pagine JSP quando eseguite dovranno produrre un documento conforme allo standard HTML5. Il codice java presente nelle JSP deve aderire alle convenzioni descritte precedentemente.

- Il tag di apertura (**<%**) dovrà essere seguito da un invio a capo;
- Il tag di chiusura (**%>**) dovrà trovarsi all'inizio della riga;
- Il codice tra i tag dovrà essere indentato;
- Nel caso di singola istruzione le tre regole precedenti possono essere evitate;

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1""
    pageEncoding="ISO-8859-1"%>
<%String stringa="prova"; %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

</body>
</html>
```

Pagine HTML

Le pagine HTML devono essere conformi allo standard HTML5 e il codice deve essere indentato, per facilitare lettura e modifiche, secondo le seguenti regole:

- Un'indentazione consiste in una tabulazione;
- Ogni tag deve essere 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 applicate ai tag normali;
- Per i tag che contengono breve testo è possibile mantenere tag di apertura e chiusura sulla stessa riga;
- Lo stile delle pagine deve essere impostato da fogli di stile CSS esterni;
- Gli script dovranno essere importati, a meno che non siano script di poche righe;

```html
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
    <div>
        <h1>Prova</h1>
        <!-- Questo
            è un commento
        -->
    </div>
</body>
</html>
```

Script

Gli script dovranno essere scritti in JavaScript o in JQuery, dovranno essere ben indentati, di facile lettura e commentati.
I nomi dei file degli script dovranno seguire la notazione “camelCase”

Fogli di stile CSS

I fogli di stile dovranno essere formattati come segue:

- I selettori della regola dovranno trovarsi sulla stessa riga;
- L’ultimo selettore della regola è seguito dalla parentesi graffa aperta “{“;
- Le proprietà che costituiscono la regola saranno una per riga e sono indentate rispetto ai selettori;
- La regola è terminata da una graffa chiusa “}” collocata da sola su una sola riga;

```css
@charset "ISO-8859-1";

html{
    background-color: red;
}

h1,h2,h3{
    color:purple;
}
```

Database SQL

- I costrutti sql devono essere scritti con sole lettere maiuscole

- I nomi delle tabelle devono essere costituiti da solo lettere minuscole, le parole devono essere separate dal carattere underscore “_”, come nel caso di tabelle generate da relazioni N-M.
- I nomi devono appartenere al dominio del problema ed esplicare correttamente ciò che intendono rappresentare.
- I nomi delle colonne delle tabelle devono seguire la notazione “camelCase”, anche loro devono esplicare correttamente la parte del dominio del problema che intendono rappresentare.

Design Pattern

MVC

Il design pattern MVC consente la suddivisione del sistema in tre blocchi principali: Model, View e Controller. Il Model modella i dati del dominio applicativo e fornisce i metodi di accesso ai dati persistenti, il View si occupa della presentazione dei dati all’utente e di ricevere da quest’ultimo gli input, infine il Controller riceve i comandi dell’utente attraverso il View e modifica lo stato di quest’ultimo e del Model.

DAO(Data Access Object) Pattern

Il DAO pattern è utilizzato per il mantenimento di una rigida separazione tra le componenti Model e Controller, in questo tipo di applicazioni basate sul paradigma MVC.

Data-Access Object: classe che implementa i metodi degli oggetti che rappresenta.
Classi Bean: classi che contengono i getters/setters degli oggetti che rappresentano e saranno usati dai DAO

1.4 Riferimenti

Documento RAD_Funisa.docx
Documento SDD_Funisa.docx

2- Packages

View

Nel folder **WebContent** contiene le **JSP** per la visualizzazione delle pagine.
Le **JSP** sono divise in base a chi può accedervi:

- quelle a cui può accedervi esclusivamente il titolare sono collocate nella cartella “**titolare”;**

- quelle a cui può accedervi esclusivamente il cliente sono collocate nella cartella "**cliente";**
- quelle comuni sono collocate nel folder **WebContent**.

JSP titolare:

- **modificaCategoriaForm.jsp,** la pagina permette di recuperare i dati inseriti nella categoria,pre compilare la form e modificare il campo desiderato;
- **modificaPerifericaForm.jsp**, la pagina permette di recuperare i dati inseriti nella periferica,pre compilare la form e modificare il campo desiderato;

JSP cliente:

- **confermaRegistrazione.jsp,** la pagina permette all'utente appena registrato di inserire il codice univoco ricevuto al momento della registrazione sulla mail personale;
- **recensisci.jsp,** la pagina consente all'utente di scrivere una recensione e dare una valutazione;
- **riepilogo.jsp,** la pagina permette all'utente di poter controllare il riepilogo della prenotazione fatta e nel caso accettarla;
- **segnalazione.jsp,** la pagina consente all'utente di poter scrivere una segnalazione nel caso non fosse soddisfatto;

JSP comuni:

- **cambioPassword.jsp,** la pagina consente all'utente di inserire la nuova password e confermarla
- **confermaRecuperoPassword.jsp,** la pagina consente all'utente registrato di inserire il codice di verifica inviato alla sua email in modo tale da recuperare la password
- **dettagliCategoria.jsp;** la pagina mostra il prodotto selezionato con nome, descrizione e immagine con la possibilità di poter scegliere delle periferiche associate
- **footer.jsp,** la pagina mostra vari link,il logo e collegamenti per i pulsanti social network;
- **header.jsp,** la pagina mostra il menu navigazionale che permette all'utente di spostarsi nella web app;
- **index.jsp,** è la pagina principale della webapp dove compaiono header ,footer e lista delle recensioni;
- **login.jsp,** questa pagina permette agli utenti di loggarsi attraverso email e password ma permette anche di recuperare la password dimenticata o accedere alla pagina di registrazione
- **notifiche.jsp,** è la pagina che permette la visualizzazione delle notifiche per l'utente
- **prenota.jsp,** questa pagina consente di cercare una postazione in base a nome,data e fascia oraria e di visualizzare il prodotto disponibile;
- **registrazione.jsp;** è la pagina che permette ad un utente non registrato di registrarsi alla piattaforma inserendo nome,cognome,email,username,password e conferma password
- **richiestaRecuperoPassword.jsp;** è la pagina dove l'utente può richiedere di recuperare la password attraverso l'inserimento della propria email
- **user.jsp;** è la pagina di gestione dell'utente registrato che varia in base al ruolo (titolare,gestore,cliente).Nel caso del cliente ,esso vedrà le proprie prenotazioni e la possibilità di modificare dati personali e/o password.Nel

caso del titolare esso vedrà le varie gestioni(prenotazioni,periferiche,postazioni,utenti,notifiche).Nel caso del gestore ,esso potrà gestire recensioni e segnalazioni.

Model

- Il package **model** contiene l'interfaccia **ModelInterface** che descrive i metodi principali che dovranno avere i dao.
- Il package **connessione** contiene la classe **DriverManagerConnectionPool** che consente la connessione al database.
- Il package **utente** contiene la classe **UtenteBean** che rappresenta un utente, la classe **UtenteDAO** che rappresenta la classe che contiene i metodi utili all'interazione con il database per ciò che riguarda la figura dell'utente.
- Il package **categoria** contiene la classe **CategoriaBean** che rappresenta una categoria, la classe **CategoriaDAO** che rappresenta la classe che contiene i metodi utili all'interazione con il database per ciò che riguarda la figura della categoria.
- Il package **notifica** contiene la classe **NotificaBean** che rappresenta una notifica, la classe **NotificaDAO** che rappresenta la classe che contiene i metodi utili all'interazione con il database per ciò che riguarda la figura della notifica.
- Il package **periferica** contiene la classe **PerifericaBean** che rappresenta una periferica, la classe **PerifericaDAO** che rappresenta la classe che contiene i metodi utili all'interazione con il database per ciò che riguarda la figura della periferica.
- Il package **postazione** contiene la classe **PostazioneBean** che rappresenta una postazione, la classe **PostazioneDAO** che rappresenta la classe che contiene i metodi utili all'interazione con il database per ciò che riguarda la figura della postazione.
- Il package **recensione** contiene la classe **RecensioneBean** che rappresenta un recensione, la classe **RecensioneDAO** che rappresenta la classe che contiene i metodi utili all'interazione con il database per ciò che riguarda la figura della recensione.
- Il package **segnalazione** contiene la classe **SegnalazioneBean** che rappresenta una segnalazione, la classe **SegnalazioneDAO** che rappresenta la classe che contiene i metodi utili all'interazione con il database per ciò che riguarda la figura della segnalazione.
- Il package **servizio** contiene la classe **ConveritoreImmagine** che permette di convertire un'immagine, la classe **RandomString** che permette di generare una stringa con caratteri casuali, la classe **Utility** che permette di inviare email, la classe **Validatore** che permette di controllare la validità dei campi del sistema.

Control

- il package **gestionePostazione** contiene le classi servlet riguardanti le postazioni:
    - **AggiungiPostazione:** permette di aggiungere una postazione data una categoria di appartentenza.
    - **EliminaPostazione:** permette di eliminare una postazione dato l'id.

    - **RestituisciListaPostazioni:** restituisce tutte le postazioni presenti nel sistema tramite **JSON.**
- il package **gestioneCategoria** contiene le classi servlet riguardanti le categorie:
    - **AggiungiCategoria**: permette di aggiungere una categoria tramite i dati inseriti.
    - **CategoriaDaModificare:** mette in sessione dati per la modifica della categoria.
    - **DettagliCategoria:** mette in sessione dettagli di una categoria**.**
    - **ModificaCategoria:** permette di modificare una categoria esistente.
    - **RestituisciListaCategorie:** restituisce le categorie presenti nel sistema tramite **JSON.**
    - **RestituisciListaCategorieLibere:** restituisce le categorie aventi almeno una postazione libera tramite **JSON.**
    - **RestituisciTipiGenerici**: restituisce i tipi generici delle categorie tramite **JSON**.
- il package **gestioneNotifica** contiene le classi servlet riguardanti le notifiche:
    - **InviaNotifica**: permette al titolare di inviare una notifica a tutti i clienti.
    - **RestituisciListaNotifiche:** permette di far visualizzare le notifiche agli utenti.
- il package **gestionePeriferica** contiene le classi servlet riguardanti le periferiche:
    - **AggiungiPeriferica:** permette di aggiungere una periferica al sistema.
    - **ModificaPeriferica:** permette di modificare una periferica esistente nel sistema.
    - **PerifericaDaModificare:** restituisce una periferica per il precompilamento del form di modifica tramite **JSON.**
    - **RestituisciListaPeriferiche:** restituisce la lista di tutte le periferiche presenti nel sistema tramite **JSON.**
    - **RestituisciListaPerifericheLibere:** restituisce la lista di tutte le periferiche dati gli input dell'utente tramite **JSON.**
    - **RestituisciPeriferica:** restituisce una periferica tramite **JSON.**
- il package **gestionePrenotazione** contiene le classi servlet riguardanti le prenotazioni:
    - **Prenota:** permette di prenotare una postazione.
    - **PrenotaIntermedia:** permette di generare un riepilogo per l'utente.
    - **RestituisciListaPrenotazioni:** restituisce la lista di tutte le prenotazioni presenti nel sistema tramite **JSON.**
- il package **gestioneRecensione** contiene le classi servlet riguardanti le recensioni:
    - **AggiungiRecensione:** permette di aggiungere una recensione al sistema.
- il package **gestioneSegnalazione** contiene le classi servlet riguardanti le segnalazioni:
    - **AggiungiSegnalazione:** permette di aggiungere una segnalazione.
- il package **gestioneUtente** contiene le classi servlet riguardanti l'utente:
    - **ConfermaRecuperoPassword:** controlla il codice dell'utente per fargli cambiare password.
    - **EliminaGestore:** permette al titolare di eliminare un gestore.
    - **Login:** permette di effettuare il login.
    - **Logout:** permette di effettuare il logout.
    - **ModificaDatiPersonali:** permette di modificare i dati personali.
    - **ModificaPassword:** permette di modificare la password.

    - **NuovaPassword:** permette di inserire una nuova password.
    - **Registrazione:** permette di effettuare la registrazione al sistema.
    - **RestituisciListaGestori:** restituisce la lista di gestori tramite **JSON.**
    - **RestituisciListaUtenti:** restituisce la lista degli utenti tramite **JSON.**
    - **RichiestaRecuperoPassword:** permette di effettuare la richiesta di recupero password.
    - **VerificaUtente:** permette all'utente di inserire il codice di conferma account.
- il package **filtro** contiene la classe **Filtro** che permette il filtraggio delle **JSP.**

TEST

Questo package contiene i test di unità del sistema in due package e la test suite:

- **AllTests:** è la test suite.
- **control:** è il package che contiene i test delle servlet seguendo la tecnica **blackbox**
- **model**: è il package che contiene i test dei dao.

Interfacce di classe

Nome Classe	AggiungiCategoria
Descrizione	Permette di aggiungere una categoria
Pre-condizione:	request.getPart("immagine")!=null&& request.getParameter("nome")!=null&& request.getParameter("tipoGenerico")!=null&& request.getParameter("descrizione")!=null&& request.getParameter("prezzo")!=null;
Post-condizione	InserisciCategoria::doPost(request,response);

Nome Classe	CategoriaDaModificare
Descrizione	Mette in sessione dati per la categoria
Pre-condizione:	
Post-condizione	CategoriaDaModificare::doPost(request,response); session.getAttribute("categoria")!=null&& session.getAttribute("nomeCategoria")!=null;;

Nome Classe	DettagliCategoria

Descrizione	Mette in sessione dati di una categoria
Pre-condizione:	
Post-condizione	DettagliCategoria::doPost(request,response); session.getAttribute(“categoria”)!=null&& session.getAttribute(“nomeCategoria”)!=null;;

Nome Classe	ModificaCategoria
Descrizione	Permette di modificare una categoria
Pre-condizione:	request.getPart(“immagine”)!=null&& request.getParameter(“nome”)!=null&& request.getParameter(“tipoGenerico”)!=null&& request.getParameter(“descrizione”)!=null&& request.getParameter(“prezzo”)!=null;
Post-condizione	ModificaCategoria::doPost(request,response);

Nome Classe	RestituisciListaCategorie
Descrizione	Ritorna una lista di categorie tramite JSON
Pre-condizione:	
Post-condizione	RestituisciListaCategorie::doPost(request,response);

Nome Classe	RestituisciListaCategorieLibere
Descrizione	Ritorna una lista di categorie aventi postazioni libere dati data,fasciaOraria e tipoGenerico tramite JSON
Pre-condizione:	request.getParameter(“data”)!=null&& request.getParameter(“fasciaOraria”)!=null&& request.getParameter(“tipoGenerico”)!=null;
Post-condizione	RestituisciListaCategorieLibere::doPost(request,response);

Nome Classe	RestituisciTipiGenerici
Descrizione	Ritorna una lista di tipi generici riferiti alle categorie tramite JSON

Pre-condizione:	
Post-condizione	RestituisciTipiGenerici::doPost(request,response);

Nome Classe	InviaNotifica
Descrizione	Permette di aggiungere una notifica
Pre-condizione:	request.getParameter(“descrizione”)!=null;
Post-condizione	InviaNotifica::doPost(request,response);

Nome Classe	AggiungiPeriferica
Descrizione	Permette di aggiungere una periferica
Pre-condizione:	request.getParameter(“nome”)!=null&& request.getParameter(“tipo”)!=null&& request.getParameter(“quantita”)!=null&& request.getParameter(“prezzo”)!=null;
Post-condizione	AggiungiPeriferica::doPost(request,response);

Nome Classe	ModificaPeriferica
Descrizione	Permette di modificare una periferica
Pre-condizione:	request.getParameter(“nome”)!=null&& request.getParameter(“tipo”)!=null&& request.getParameter(“quantita”)!=null&& request.getParameter(“prezzo”)!=null;
Post-condizione	ModificaPeriferica::doPost(request,response);

Nome Classe	PerifericaDaModificare
Descrizione	Restituisce una periferica per il precompilamento del form

	tramite JSON
Pre-condizione:	
Post-condizione	PerifericaDaModificare::doPost(request,response); session.getAttribute("periferica")!=null&& session.getAttribute("nomePeriferica");

Nome Classe	RestituisciListaPeriferiche
Descrizione	Restituisce una lista contente tutte le periferiche del sistema tramite JSON
Pre-condizione:	
Post-condizione	RestituisciListaPeriferiche::doPost(request,response);

Nome Classe	RestituisciListaPerifericheLibere
Descrizione	Restituisce una lista contente tutte le periferiche libere del sistema tramite JSON
Pre-condizione:	session.getAttribute("data")!=null&& session.getAttribute("fasciaOraria")!=null;
Post-condizione	RestituisciListaPerifericheLibere::doPost(request,response);

Nome Classe	RestituisciPeriferica
Descrizione	Restituisce una periferica del sistema tramite JSON
Pre-condizione:	
Post-condizione	RestituisciPeriferica::doPost(request,response);

Nome Classe	AggiungiPostazione
Descrizione	Permette di aggiungere una postazione al sistema

Pre-condizione:	request.getParameter(“nomeCategoria”)!=null;
Post-condizione	AggiungiPostazione::doPost(request,response);

Nome Classe	EliminaPostazione
Descrizione	Permette di eliminare una postazione al sistema
Pre-condizione:	request.getParameter(“id”)!=null;
Post-condizione	EliminaPostazione::doPost(request,response);

Nome Classe	RestituisciListaPostazioni
Descrizione	Restituisce le postazioni del sistema tramite JSON.
Pre-condizione:	
Post-condizione	RestituisciListaPostazioni::doPost(request,response);

Nome Classe	Prenota
Descrizione	Permette di prenotare una postazione
Pre-condizione:	session.getAttribute(“datiPrenotazione”)!=null&& session.getAttribute(“datiPeriferiche”)!=null;
Post-condizione	Prenota::doPost(request,response);

Nome Classe	PrenotaIntermedia
Descrizione	Permette di generare un riepilogo della prenotazione per l’utente
Pre-condizione:	
Post-condizione	PrenotaIntermedia::doPost(request,response);

Nome Classe	RestituisciListaPrenotazioni
Descrizione	Ritorna la lista delle prenotazioni del sistema tramite JSON
Pre-condizione:	
Post-condizione	RestituisciListaPrenotazioni::doPost(request,response);

Nome Classe	AggiungiRecensioni
Descrizione	Permette di aggiungere una recensione
Pre-condizione:	request.getParameter(“descrizione”)!=null&& request.getParameter(“valutazione”)!=null;
Post-condizione	AggiungiRecensioni::doPost(request,response);

Nome Classe	AggiungiSegnalazione
Descrizione	Permette di aggiungere una segnalazione
Pre-condizione:	request.getParameter(“descrizione”)!=null&& request.getParameter(“tipo”)!=null;
Post-condizione	AggiungiSegnalazione::doPost(request,response);

Nome Classe	ConfermaRecuperoPassword
Descrizione	Controlla il codice dell’utente per fargli cambiare password
Pre-condizione:	request.getParameter(“codiceVerifica”)!=null&& session.getAttribute(“emailCodice”)!=null;
Post-condizione	ConfermaRecuperoPassword::doPost(request,response);

Nome Classe	EliminaGestore
Descrizione	Permette di eliminare un gestore
Pre-condizione:	

Post-condizione	EliminaGestore::doPost(request,response);

Nome Classe	Login
Descrizione	Permette di effettuare il login
Pre-condizione:	request.getParameter(“email”)!=null&& request.getParameter(“password”)!=null;
Post-condizione	Login::doPost(request,response); session.getAttribute(“utente”)!=null;

Nome Classe	Logout
Descrizione	Permette di effettuare il logout
Pre-condizione:	
Post-condizione	Logout::doPost(request,response); session==null;

Nome Classe	ModificaDatiPersonali
Descrizione	Permette di modificare i dati personali
Pre-condizione:	request.getParameter(“nome”)!=null&& request.getParameter(“cognome”)!=null&& request.getParameter(“username”)!=null;
Post-condizione	ModificaDatiPersonali::doPost(request,response);

Nome Classe	ModificaPassword
Descrizione	Permette di modificare la password
Pre-condizione:	request.getParameter(“vecchiaPassword”)!=null&& request.getParameter(“nuovaPassword”)!=null&& request.getParameter(“confermaPassword”)!=null&& session.getAttribute(“utente”)!=null;
Post-condizione	ModificaPassword::doPost(request,response);

Nome Classe	NuovaPassword
Descrizione	Permette di inserire una nuova password con il recupero
Pre-condizione:	request.getParameter(“Password”)!=null&& request.getParameter(“confermaPassword”)!=null&& session.getAttribute(“emailCodice”)!=null;
Post-condizione	NuovaPassword::doPost(request,response);

Nome Classe	Registrazione
Descrizione	Permette di inserire una nuova password con il recupero
Pre-condizione:	request.getParameter(“nome”)!=null&& request.getParameter(“cognome”)!=null&& request.getParameter(“email”)!=null&& request.getParameter(“username”)!=null&& request.getParameter(“password”)!=null&& request.getParameter(“confermaPassword”)!=null;
Post-condizione	Registrazione::doPost(request,response);

Nome Classe	RestituisciListaGestori
Descrizione	Ritorna una lista contente tutti i gestori del sistema tramite JSON
Pre-condizione:	
Post-condizione	RestituisciListaGestori::doPost(request,response);

Nome Classe	RestituisciListaUtenti
Descrizione	Ritorna una lista contente tutti gli utenti del sistema tramite JSON
Pre-condizione:	

Post-condizione	RestituisciListaUtenti::doPost(request,response);

Nome Classe	RichiestaRecuperoPassword
Descrizione	Permette di avviare la procedura di recupero password
Pre-condizione:	
Post-condizione	RestituisciListaGestori::doPost(request,response);

Nome Classe	VerificaUtente
Descrizione	Permette di verificare l'utente
Pre-condizione:	request.getParameter("codiceVerifica")!=null&& session.getAttribute("utente")!=null;
Post-condizione	VerificaUtente::doPost(request,response);

3-ClassDiagram