ODD(Object Design Document)

Smart Cooking

Riferimento	
Versione	0.2
Data	08/11/2021
Destinatario	Prof.ssa F. Ferrucci
Presentato da	Benedetta del Giudice, Daniele Colucci, Marco Luisi, Antonio Alfano, Anton Zolotko
Approvato da	

Revision History

Data	Versione	Descrizione	Autori
08/09/2021	0.1	Prima stesura	Benedetta del Giudice
18/10/2021	0.2	Revisione	Benedetta del Giudice, Daniele Colucci, Marco Luisi, Antonio Alfano, Anton Zolotko

Sommario

1. Introduzione

1.1 Object Design trade-offs

Nella fase dell'Object Design si va a dettagliare quanto individuato nell'analisi dei requisiti e a prendere decisioni implementative che permettono di avvicinarsi maggiormente agli obiettivi preposti, scegliendo tra le soluzioni possibili. In particolare sono stati individuati i seguenti trade-off:

Memoria vs Estensibilità: Il sistema dovrà essere dotato di una quantità di memoria crescente nel tempo, a causa dello spazio richiesto dai file relativi alle videolezioni aggiunte. Come conseguenza, sarà disponibile un margine di memoria che permetterà di sviluppare nuove funzionalità senza avere estremi vincoli.

Tempo di risposta vs Affidabilità: Il sistema metterà in primo piano l'affidabilità controllando in maniera accurata i dati immessi in input ma senza trascurare totalmente i tempi di risposta.

Disponibilità vs Tolleranza ai guasti: Il sistema, in caso si verifichino errori in una funzionalità, notifica l'utente del malfunzionamento, ma permette di rieffettuare l'azione eseguita.

Criteri di manutenzione vs Criteri di performance: Il sistema sarà implementato preferendo la manutenibilità alla performance in modo da facilitare gli sviluppatori nel processo di aggiornamento del software a discapito delle performance del sistema.

1.1.1 Componenti off-the-shelf

Per il nostro sistema utilizzeremo componenti off-the-shelf, cioè componenti software già disponibili utilizzati per facilitare e velocizzare la creazione del software. Il framework che utilizzeremo per il comparto grafico è Bootstrap, un framework open source adatto a creare la grafica di siti web e web application. Questo contiene modelli di progettazione basati su HTML e CSS sia per la tipografia che per le varie componenti dell'interfaccia (moduli, pulsanti e navigazione) così come alcune estensioni opzionali in Javascript.

Il back-end farà invece forte affidamento sulle Servlet, componenti software di Java che operano all'interno di un server web, permettendo la creazione di applicazioni web.

Il front-end sarà realizzato dinamicamente tramite JavaServer Pages, le quali permettono la comunicazione con il back-end per fornire informazioni aggiornate e rilevanti.

Per la realizzazione di alcune funzionalità interattive utilizzeremo la tecnica AJAX e la libreria jQuery.

1.1.2 Design patterns

I design patterns sono template di soluzioni a problemi ricorrenti impiegati per ottenere riuso e flessibilità.

Data Access Object

Nome: Data Access Object

Descrizione del problema: Aggiungere uno strato di astrazione tra la logica di business e le query relative ai dati persistenti.

Soluzione: Separare un oggetto nella logica di business e i suoi dati persistenti sfruttando una classe pattern connessa con il database, la quale implementa dei metodi comunicanti con le tabelle persistenti tramite query. La classe pattern andrà dunque a effettuare le operazioni di conversione e dunque comunicare i risultati della query all'oggetto. La tecnologia usata è anche parzialmente compatibile con altri tipi di database, in modo da dover richiedere meno sforzo possibile in caso di cambio tecnologia di gestione dei dati persistenti.

Conseguenze: Il pattern DAO consente ai meccanismi di accesso ai dati di cambiare indipendentemente dal codice che utilizza i dati. Il vantaggio relativo all'uso del DAO è dunque il mantenimento di una rigida separazione tra le componenti di un'applicazione, le quali potrebbero essere il "Model" e il "Control" in un'applicazione basata sul paradigma MVC.

Repository

Nome: Repository

Descrizione del problema: Creare un'interfaccia per facilitare la gestione delle collezioni di oggetti.

Soluzione: Implementare una classe che sfrutta i singoli oggetti allo scopo di creare collezioni. Nello specifico, sfruttare le classi singole DAO per separare il codice della creazione dei singoli oggetti, e creare la collezione relativa tramite l'interfaccia Repository.

Conseguenze: Permette la creazione di un ulteriore layer di astrazione tra la logica di business e l'implementazione della comunicazione con i dati persistenti. Rende inoltre più facile la scrittura e la leggibilità del codice, permettendo di concentrarsi sulla implementazione dei requisiti anziché sull'interazione con il database.

Model-View-Controller

Nome: Model-View-Controller

Descrizione del problema: Separare la logica dei dati di business dalla logica di presentazione.

Soluzione: Suddividere l'implementazione in tre strati: strato relativo alla logica dei dati persistenti (Model), strato relativo alla logica dei dati di presentazione (View) e strato relativo alla logica di business (Controller).

Conseguenze: Miglioramento della gerarchia della struttura dell'architettura del software. Astrazione degli strati con funzioni non condivise.

1.2 Linee Guida per la Documentazione delle interfacce

Nell'implementazione del sistema, i programmatori dovranno attenersi alle linee guida di seguito definite.

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

```
<!- Accettabile ->
<div>
      <span>
            <ul>
                  <li>
                      Uno
                  </li>
                  <li>
                      Due
                  </li>
            </ul>
      </span>
</div>

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

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:

```
/**
* 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   fX (a, b, c) {
```

Fogli di stile CSS

I fogli di stile (CSS) devono seguire le seguenti convenzioni:

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

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 Definizioni, Acronimi e abbreviazioni

Off-The-Shelf: Servizi esterni al sistema di cui viene fatto utilizzo.

Bootstrap: è una raccolta di strumenti liberi per la creazione di front-end relativo a siti e applicazioni per il web.

HTML: Linguaggio di markup utilizzato per lo sviluppo di pagine Web.

CSS: acronimo di Cascading Style Sheets, è un linguaggio usato per definire la formattazione di pagine HTML e relativi linguaggi.

JavaScript: JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione di effetti dinamici interattivi ed eventi.

jQuery: jQuery è una libreria JavaScript per semplificare lo scripting degli elementi nei documenti HTML, e offre un'interfaccia semplificata per la creazione di richieste e gestione di risposte da effettuare tramite AJAX.

AJAX: AJAX, acronimo di Asynchronous JavaScript and XML, è una tecnica di sviluppo software per la realizzazione di applicazioni web interattive.

Tomcat: Server web che gestisce un'applicazione web tramite Servlet e pagine dinamiche scritte in JavaServer Pages.

Servlet: Oggetti scritti in linguaggio Java, che operano all'interno di un server web (Tomcat), che forniscono funzioni all'applicazione web.

JavaServer Pages (JSP): Tecnologia utilizzata per fornire contenuti dinamici in front-end. Si basa su HTML e Java, e permette l'utilizzo di linguaggi di scripting quali JavaScript. Vengono tradotte in Servlet in compilazione.

1.4 Riferimenti

NC02_RAD_V2.2

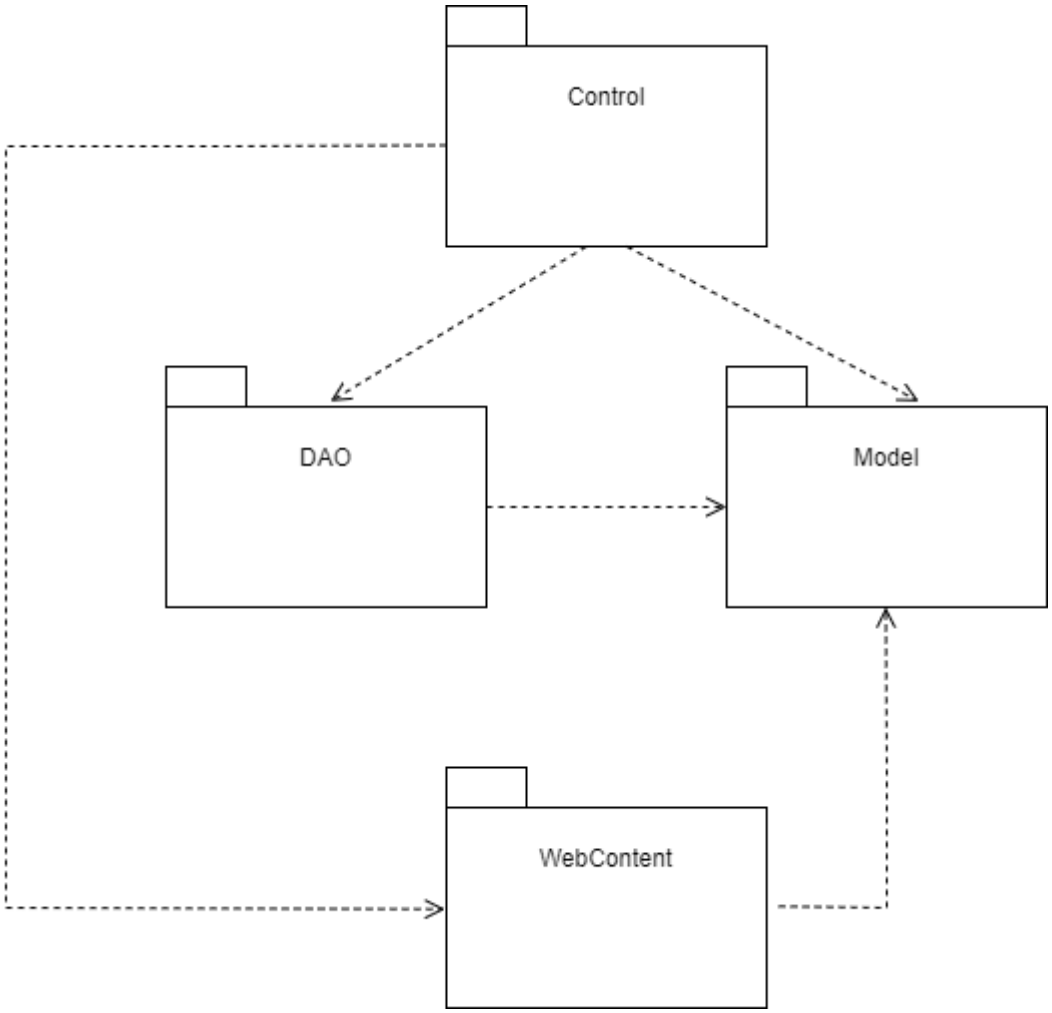
NC02_SDD_V0.4

Libro:

-- Object-Oriented Software Engineering (Conquering Complex and Changing Systems) Autori:

-- Bernd Bruegge & Allen H. Dutoit

2. Packages



Nome package	Control
Descrizione	Definisce le Servlet che ricevono i comandi dell'utente, attraverso la View rappresentata dalle JSP definite all'interno del pacchetto WebContent, e li attuano modificando gli oggetti del dominio della soluzione e la View
Dipendenze	DAO, Model, WebContent

Nome package	Model
Descrizione	Definisce le classi che modellano gli oggetti del dominio della soluzione.
Dipendenze	

Nome package	DAO
Descrizione	Definisce le classi connesse con il database e che implementano i metodi comunicanti con le tabelle persistenti tramite query. Tali classi andranno ad effettuare le operazioni di conversione e dunque comunicare i risultati della query agli oggetti del dominio della soluzione.
Dipendenze	Model

Nome package	WebContent
Descrizione	Definisce le JSP che visualizzano i dati relativi agli oggetti del dominio della soluzione e che si occupano dell'interazione con gli utenti.
Dipendenze	Model

3. Class interfaces

Nome classe	Registrazione
Descrizione	Consente la registrazione di un nuovo utente al sistema
Pre-condizione	(patternNome.matcher(nome).find() && nome.length!=0) && (patternCognome.matcher(cognome).find() && cognome.length!=0) && passoword.length()>=8 && (patternEmail.matcher(email).find() && email.length!=0) && (patternTelefono.matcher(telefono).find() && telefono.length!=0) && (patternDataNascita.matcher(dataNascita).find() && dataNascita.length!=0)
Post-condizione	request.getRequestDispatcher("utente.jsp").forward(request, response);
Invarianti	

Nome classe	Abbonamento
Descrizione	Consente ad un utente registrato al sistema di sottoscrivere un abbonamento

Pre-condizione	utenteRepository.getUtente(email)!=null && (durata.equals("mensile") || durata.equals("annuale"))
Post-condizione	
Invarianti	

Nome classe	ModificaDatiUtente
Descrizione	Consente ad un utente registrato al sistema di poter modificare i propri dati personali.
Pre-condizione	utenteRepository.getUtente(email)!=null && (patternNome.matcher(nome).find() && nome.length!=0) && (patternCognome.matcher(cognome).find() && cognome.length!=0) && passoword.length()>=8 && (patternEmail.matcher(email).find() && email.length!=0) && (patternTelefono.matcher(telefono).find() && telefono.length!=0) && (patternDataNascita.matcher(dataNascita).find() && dataNascita.length!=0)
Post-condizione	
Invarianti	

Nome classe	VideoCategoria
Descrizione	Restituisce, tramite JSON, la lista delle videolezioni in base alla categoria presa in input
Pre-condizione	val.equals(Antipasti) || val.equals("Primi") || val.equals("Secondi") || val.equals("Contorni") || !val.equals("Dolci")
Post-condizione	response.setContentType("application/json");
Invarianti	

Nome classe	VideoKeyword
Descrizione	Restituisce, tramite JSON, la lista delle videolezioni in base alla parola chiave presa in input

Pre-condizione	val!=null
Post-condizione	response.setContentType("application/json");
Invarianti	

Nome classe	CaricamentoVideo
Descrizione	Consente ad un amministratore di caricare una videolezione all'interno del sistema
Pre-condizione	(patternTitolo.matcher(titolo).find()) && titolo.length()!=0) && (tipologia.equals("true") || tipologia.equals("false") && (categoria.equals("Antipasti") || categoria.equals("Primi") || categoria.equals("Secondi") || categoria.equals("Dolci") || categoria.equals("Contorni")) && (idInsegnante ==1 || idInsegnante ==2 || idInsegnante ==3 || idInsegnante ==4 || idInsegnante ==5 || idInsegnante ==6) && (email_admin.equals("paoloRossi80@gmail.com") || email_admin.equals("micheleBianchi77@gmail.com")
Post-condizione	request.getRequestDispatcher("admin.jsp").forward(request, response);
Invarianti	

Nome classe	VisualizzazioneListaUtenti
Descrizione	Restituisce, tramite JSON, la lista di tutti gli utenti registrati al sistema
Pre-condizione	
Post-condizione	response.setContentType("application/json");
Invarianti	

Nome classe	VisualizzazioneListaVideo

Descrizione	Restituisce, tramite JSON, la lista di tutte le videolezioni presenti all'interno del sistema
Pre-condizione	
Post-condizione	response.setContentType("application/json");
Invarianti	

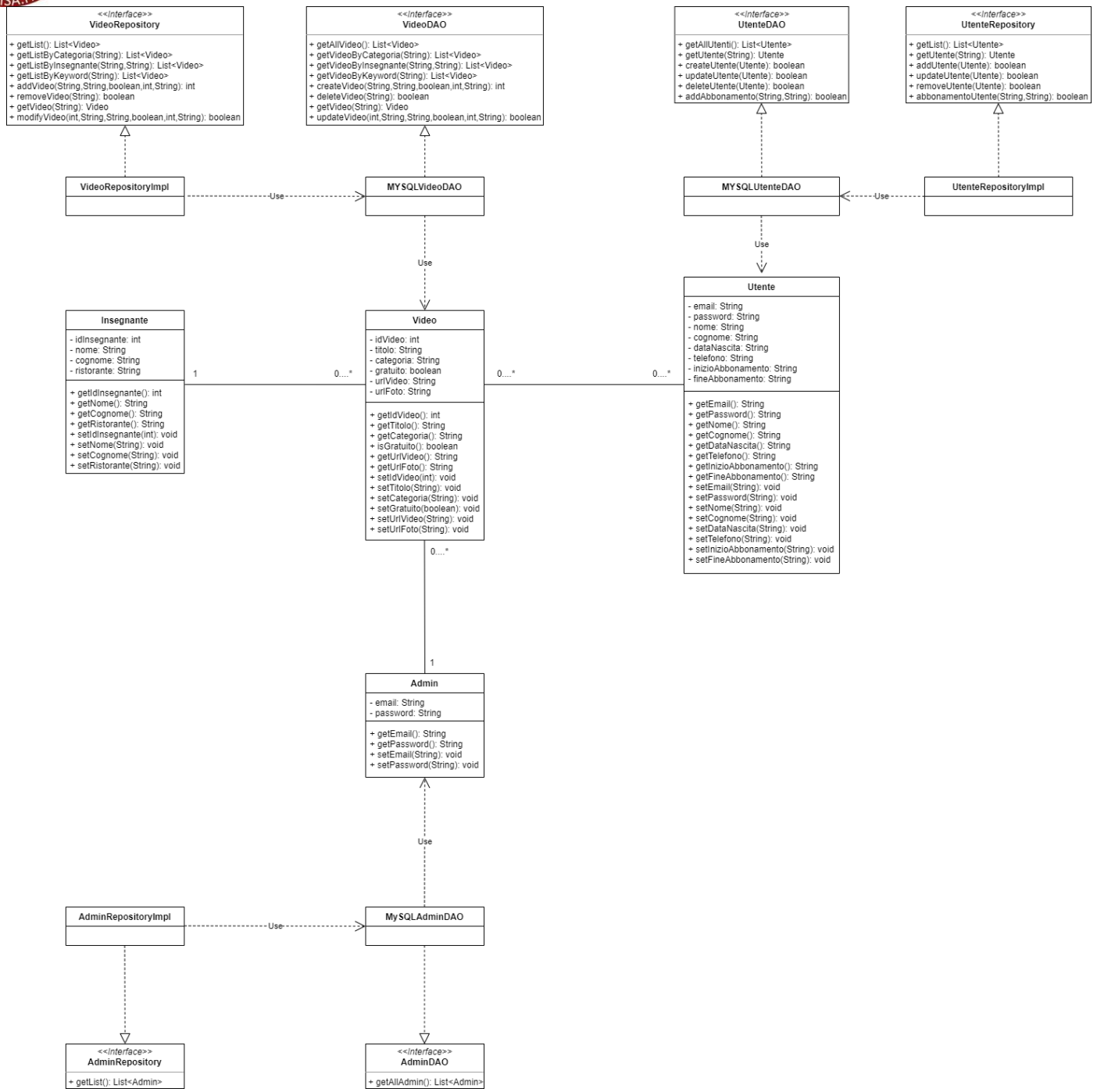
Nome classe	RemoveVideoAdmin
Descrizione	Consente di eliminare una videolezione dal sistema
Pre-condizione	videoRepository.getVideo(titolo)!=null
Post-condizione	
Invarianti	

Nome classe	UpdateVideoAdmin
Descrizione	Consente di modificare i dati di una videolezione presente all'interno del sistema
Pre-condizione	(patternTitolo.matcher(titolo).find()) && titolo.length()!=0) && (tipologia.equals("true") || tipologia.equals("false") && (categoria.equals("Antipasti") || categoria.equals("Primi") || categoria.equals("Secondi") || categoria.equals("Dolci") || categoria.equals("Contorni")) && (idInsegnante ==1 || idInsegnante ==2 || idInsegnante ==3 || idInsegnante ==4 || idInsegnante ==5 || idInsegnante ==6) && (email_admin.equals("paoloRossi80@gmail.com") || email_admin.equals("micheleBianchi77@gmail.com")
Post-condizione	
Invarianti	

Nome classe	Login
Descrizione	Consente di effettuare il login al sistema
Pre-condizione	
Post-condizione	session.setAttribute("user", user); session.setAttribute("admin", admin);
Invarianti	

Nome classe	Logout
Descrizione	Consente di effettuare il logout dal sistema
Pre-condizione	
Post-condizione	session.invalidate(); request.getRequestDispatcher("login.jsp").forward(request, response);
Invarianti	

4. Class diagram


<<Interface>>
VideoRepository
+ getList(): List<Video>
+ getListByCategoria(String): List<Video>
+ getListByInsegnante(String,String): List<Video>
+ getListByKeyword(String): List<Video>
+ addVideo(String,String,boolean,int,String): int
+ removeVideo(String): boolean
+ getVideo(String): Video
+ modifyVideo(int,String,String,boolean,int,String): boolean
<<Interface>>
VideoDAO
+ getAllVideo(): List<Video>
+ getVideoByCategoria(String): List<Video>
+ getVideoByInsegnante(String,String): List<Video>
+ getVideoByKeyword(String): List<Video>
+ createVideo(String,String,boolean,int,String): int
+ deleteVideo(String): boolean
+ getVideo(String): Video
+ updateVideo(int,String,String,boolean,int,String): boolean
<<Interface>>
UtenteDAO
+ getAllUtenti(): List<Utente>
+ getUtente(String): Utente
+ createUtente(Utente): boolean
+ updateUtente(Utente): boolean
+ deleteUtente(Utente): boolean
+ addAbbonamento(String,String): boolean
<<Interface>>
UtenteRepository
+ getList(): List<Utente>
+ getUtente(String): Utente
+ addUtente(Utente): boolean
+ updateUtente(Utente): boolean
+ removeUtente(Utente): boolean
+ abbonamentoUtente(String,String): boolean
VideoRepositoryImpl
Use
MYSQLVideoDAO
MYSQLUtenteDAO
Use
UtenteRepositoryImpl
Use
Use
Insegnante
- idInsegnante: int
- nome: String
- cognome: String
- ristorante: String
+ getIdInsegnante(): int
+ getNome(): String
+ getCognome(): String
+ getRistorante(): String
+ setIdInsegnante(int): void
+ setNome(String): void
+ setCognome(String): void
+ setRistorante(String): void
1
0....*
Video
- idVideo: int
- titolo: String
- categoria: String
- gratuito: boolean
- urlVideo: String
- urlFoto: String
+ getIdVideo(): int
+ getTitolo(): String
+ getCategoria(): String
+ isGratuito(): boolean
+ getUrlVideo(): String
+ getUrlFoto(): String
+ setIdVideo(int): void
+ setTitolo(String): void
+ setCategoria(String): void
+ setGratuito(boolean): void
+ setUrlVideo(String): void
+ setUrlFoto(String): void
0....*
0....*
Utente
- email: String
- password: String
- nome: String
- cognome: String
- dataNascita: String
- telefono: String
- inizioAbbonamento: String
- fineAbbonamento: String
+ getEmail(): String
+ getPassword(): String
+ getNome(): String
+ getCognome(): String
+ getDataNascita(): String
+ getTelefono(): String
+ getInizioAbbonamento(): String
+ getFineAbbonamento(): String
+ setEmail(String): void
+ setPassword(String): void
+ setNome(String): void
+ setCognome(String): void
+ setDataNascita(String): void
+ setTelefono(String): void
+ setInizioAbbonamento(String): void
+ setFineAbbonamento(String): void
0....*
1
Admin
- email: String
- password: String
+ getEmail(): String
+ getPassword(): String
+ setEmail(String): void
+ setPassword(String): void
Use
AdminRepositoryImpl
Use
MySQLAdminDAO
<<Interface>>
AdminRepository
+ getList(): List<Admin>
<<Interface>>
AdminDAO
+ getAllAdmin(): List<Admin>

5. Glossario

Videolezione si riferisce al video caricato dall'amministratore e visualizzabile dall'utente. La visualizzazione delle videolezioni può essere gratuita o richiedere un abbonamento.

Utente si riferisce a chiunque acceda al sito con lo scopo di usufruire dei servizi del sistema quali visualizzare una videolezione. Non è specificato lo stato di autenticazione.

Amministratore si riferisce alle persone designate a gestire i contenuti della piattaforma. Non è utilizzato per indicare chi si occupa della manutenzione della piattaforma.