**Progetto OOP – Deliverables 2015-2016**

**“Object Oriented Programming”**

**a.a. 2015-2016**

|  |
| --- |
| **Biblioteca Virtuale** |

**TEAM:**

|  |  |  |
| --- | --- | --- |
| Nome & Cognome | Matricola | E-mail address: |
| Leonardo Ruggeri | 231254 | leonardo.ruggeri@student.univaq.it |
| Valentina Di Cristofaro | 210932 | valedicr@student.univaq.it |
| Marco Bucci | 202443 | marckus91@gmail.com |

**Progetto OOP – Deliverables 2015-2016**

**INDICE**

**Analisi dei requisiti**

**Modello Analisi**

**Architettura software**

**Class Diagram**

**Design di progettazione**

**Progetto OOP – Deliverables 2015-2016**

**Analisi dei Requisiti  
Che cosa deve fare il sistema?**

Prima di passare all’analisi dei requisiti descriviamo brevemente cosa deve fare il sistema.

Il sistema deve gestire un insieme di collezioni, servizi e persone in modo da supportare le fasi di creazione, uso, preservazione dei dati e diffusione.

Deve essere in grado, tramite un’interfaccia grafica, di permettere ad un utente di immagazzinare immagini, opere e libri.

Le opere devono poter essere pubblicate e successivamente consultate dagli utenti.

Sono stati identificati i seguenti attori del sistema:  

* **Amministratore**: gestione generale del sistema;
* **Acquisitore**: acquisizione/digitalizzazione immagine;
* **Utente base**: visualizzazione elenco, titoli, opere;
* **Utente avanzato**: visualizzazione completa delle opere;

**Priorità dei Requisiti**

* Immissione immagini
* Pubblicazione immagini e testo
* Consultazione titoli
* Consultazione contenuti

**Progetto OOP – Deliverables 2015-2016**

I requisiti di maggior importanza sono la trascrizione del testo. Viene data minor importanza alla consultazione dei titoli e dei contenuti in quanto l’operazione all’inizio verrà poco eseguita.

* **Acquisizione immagini :** Per fare ciò è necessario acquisire immagini ad alta risoluzione degli antichi libri. L’acquisitore dovrà semplicemente caricare le immagini in formato “jpg” o “png” dei manoscritti e assegnarle all’opera di riferimento. Non c’è un limite per il numero di immagini assegnabili a un’opera.
* **Ricerca opere :** Gli utenti, a seconda che abbiano un account base o un account avanzato, potranno visualizzare l’elenco di tutte le opere presenti nel sistema o ricercare una specifica opera e leggerne l’intera trascrizione.
* **Ricerca per nome :** Gli utenti possono cercare i libri in base al titolo.
  + **Ricerca per autore :** E’ anche possibile cercare tutte le opere di un certo autore.
  + **Ricerca per editore :** Infine è possibile cercare le opere di uno specifico editore.
* **Visualizzazione opere**  Gli utenti, potranno visualizzare le opere presenti nel sistema.
* **Gestione database**  Il sistema conserva tutti i dati necessari (immagini, trascrizioni, dati utente) in un database gestito dall’amministratore di sistema.
  + **Aggiunta immagini :** L’acquisitore aggiunge le immagini dell’opera al database.
  + **Aggiunta trascrizioni :** Il trascrittore aggiunge nel database le trascrizioni da lui effettuate.
* **Gestione account**
  + **Registrazione :** Gli utenti per usare il sistema devono registrarsi, scegliendo se essere utenti base o utenti avanzati.
  + **Login:** Una volta effettuata la registrazione gli utenti devono effettuare il login per usufruire dei servizi offerti dal sistema.
  + **Cancellazione utenti :** E’ possibile cancellare gli utenti registrati o loggati.

**Consultazione contenuti**

L’utente deve poter consultare i titoli e le opere

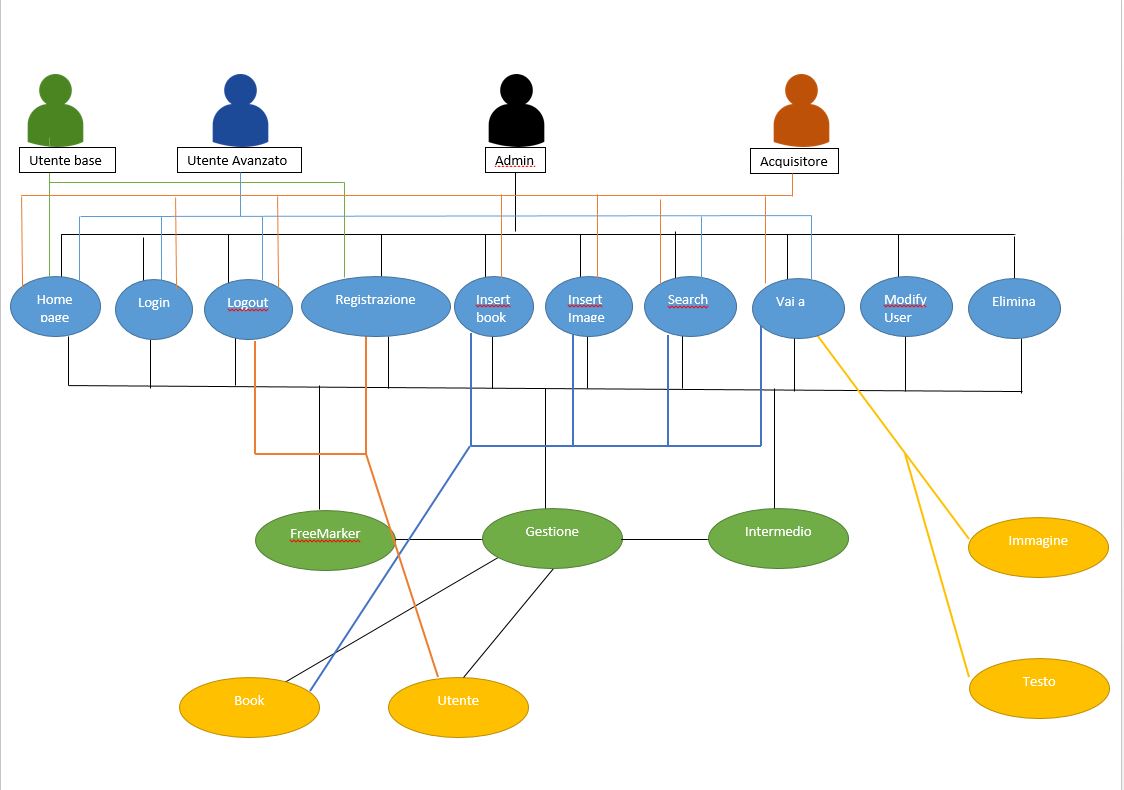
**Requisiti non funzionali**

**Usability :** il Sistema deve poter essere utilizzato facilmente da qualsiasi tipo di utente.

**Progetto OOP – Deliverables 2015-2016**

**Modello di Analisi**

Dall’ analisi dei requisiti si individuano le seguenti classi, suddivise nelle tre tipologie previste da UML:



*Classe Model*: rappresentano l’interfaccia che consente agli utenti di interagire col sistema, nel nostro caso gli utenti utilizzano molte interfacce grafiche, questo perché ogni utente deve svolgere una diversa operazione. Questo ci permetterà in seguito di poter gestire separatamente le diverse interfacce.

*Classe Servl*: pagine principali che il sistema andrà a visualizzare per utilizzare la nostra biblioteca.

*Classe Uti*: Conterranno le funzioni che consentono di fare la connessione al database, con le seguenti query in SQL e **FreeMarker** è un [Java](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&prev=search&rurl=translate.google.it&sl=en&u=https://en.wikipedia.org/wiki/Java_%28programming_language%29&usg=ALkJrhjqUzPDz7Ycfwv97vZqowmidb8grw) -based [Template Engine](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&prev=search&rurl=translate.google.it&sl=en&u=https://en.wikipedia.org/wiki/Template_engine_%28web%29&usg=ALkJrhiX5KbxKC422JcJtt7gXVU_-36N8w) , originariamente incentrata sulla generazione dinamica pagina web con [MVC](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&prev=search&rurl=translate.google.it&sl=en&u=https://en.wikipedia.org/wiki/Model-view-controller&usg=ALkJrhjPLNH1a-zMLLyW7sCUwST4glwsrg) [architettura software](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&prev=search&rurl=translate.google.it&sl=en&u=https://en.wikipedia.org/wiki/Software_architecture&usg=ALkJrhhZ1ZBv7tVsjxPfJ_0xBmGrtKZ7Pg) . Tuttavia, si tratta di un motore di template generico scopo, senza alcuna dipendenza da [servlet](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&prev=search&rurl=translate.google.it&sl=en&u=https://en.wikipedia.org/wiki/Servlet&usg=ALkJrhgyFJKaH89DBUyBQECJOAYMv79MBQ) o [HTTP](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&prev=search&rurl=translate.google.it&sl=en&u=https://en.wikipedia.org/wiki/HTTP&usg=ALkJrhhpiXTTJ4MsOYz8IVVeyJLEdrwGYQ) o [HTML](https://translate.googleusercontent.com/translate_c?depth=1&hl=it&prev=search&rurl=translate.google.it&sl=en&u=https://en.wikipedia.org/wiki/HTML&usg=ALkJrhiBhuBwHVR0yGsCnXgFc8y3pb833g) , e quindi è spesso usato per la generazione di codice sorgente, i file di configurazione o e-mail.

Queste funzionalità rappresentano ad alto livello cosa il sistema dovrà fare.

**Progetto OOP – Deliverables 2015-2016**

**Architettura Software**

Come modello architetturale è stato scelto il pattern MVC per le sue peculiarità nella separazione delle logiche, infatti un sistema è gestibile se è diviso in parti che siano indipendenti tra loro. Il pattern architetturale, nell’intento di disaccoppiare il sistema, stabilisce tre parti in cui esso verrà diviso:

* Rappresentazione del modello di dominio (model);
* Interfaccia utente (view);
* Controllo dell’interazione uomo-macchina (controller).

**IL PATTERN MVC**

****

Il **Model-View-Controller** (**MVC**, talvolta tradotto in italiano con la dicitura **Modello-Vista-Controllo**), è un [pattern architetturale](https://it.wikipedia.org/wiki/Design_pattern) molto diffuso nello sviluppo di sistemi [software](https://it.wikipedia.org/wiki/Software), in particolare nell'ambito della [programmazione orientata agli oggetti](https://it.wikipedia.org/wiki/Programmazione_orientata_agli_oggetti), in grado di separare la [logica di presentazione](https://it.wikipedia.org/wiki/Livello_di_presentazione#Altro) dei dati dalla logica di bussiness.

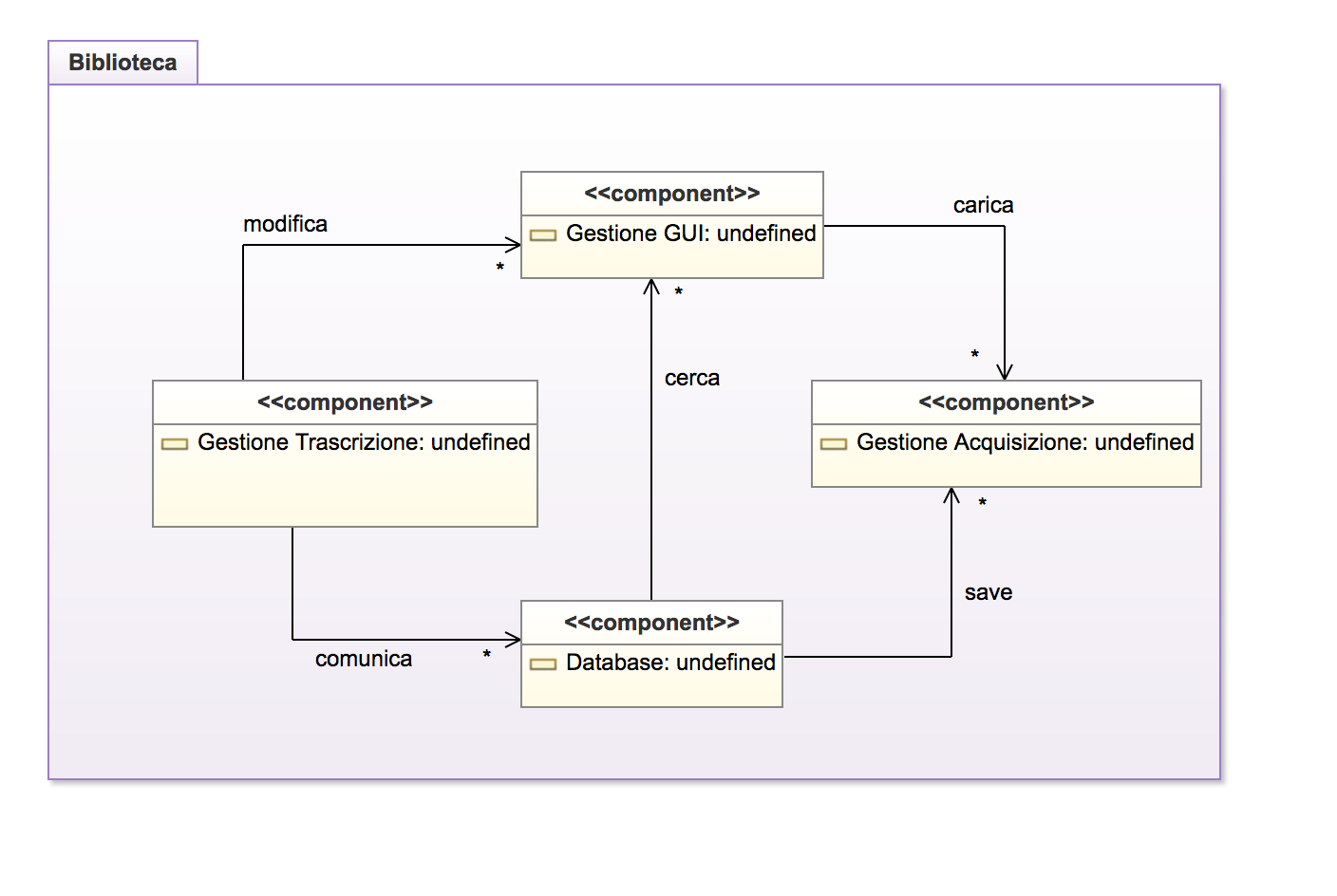
**Progetto OOP – Deliverables 2015-2016**

l pattern è basato sulla separazione dei compiti fra i componenti [software](https://it.wikipedia.org/wiki/Software) che interpretano tre ruoli principali:

* il **model** fornisce i [metodi](https://it.wikipedia.org/wiki/Metodo_%28informatica%29) per accedere ai dati utili all'applicazione;
* il **view** visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
* il **controller** riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti.
* Il componente centrale del MVC, il modello, cattura il comportamento dell'applicazione in termini di dominio del problema, indipendentemente dall'interfaccia utente.
* Il modello gestisce direttamente i dati, la logica e le regole dell'applicazione.
* Una vista può essere una qualsiasi rappresentazione in output di informazioni, come un grafico o un diagramma. Sono possibili viste multiple delle stesse informazioni, come ad esempio un grafico a barre per la gestione e la vista tabellare per l'amministrazione.
* La terza parte, il controller, accetta l'input e lo converte in comandi per il modello e/o la vista.

**Progetto OOP – Deliverables 2015-2016**

**MVC : Architettura Target**

****

Il Component Diagram in figura, ha lo scopo principale di mostrare le relazioni strutturali tra le componenti di un sistema (una componente è un elemento fisico rimpiazzabile del sistema). Le componenti coinvolte sono le seguenti:

**Gestione Gui** : Rappresenta le interfacce grafiche con le quali l'utente utilizza il sistema. Si interfaccia con il database Attraverso l’interfaccia “Search” passandogli il titolo o il codice isbn del libro il quale poi verrà mostrato all’utente che ne fa richiesta.

**Database**: Questa componente rappresenta il database che conterrà le tabelle contenenti i “libri”, composti da immagini e relative trascrizioni, e le varie informazioni degli utenti.

**Gestione Acquisizione** : Rappresenta la componente che permette la scannerizzazione del documento. Si interfaccia con Management Gui attraverso l’interfaccia “carica”, con la quale l’utente acquisitore avvia la componente e successivamente si interfaccia con il Database attraverso “salva”, passandogli l’immagine del libro la quale verrà salvata nel database.

**Gestione Trascrizione**: Rappresenta la componente che permette al Trascrittore di trascrivere il libro. Si interfaccia con il database, per richiedere l’immagine di un libro.

**Progetto OOP – Deliverables 2015-2016**

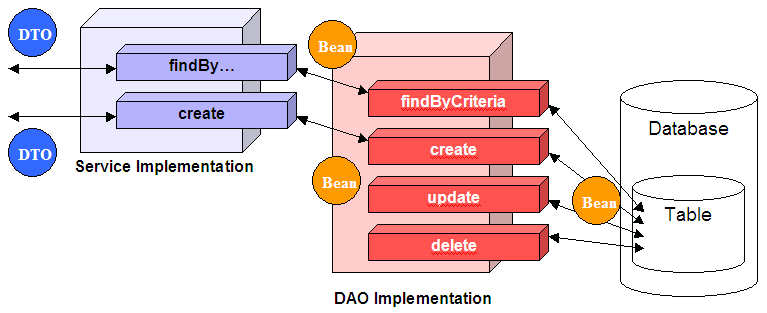
Inoltre per la gestione persistente dei dati viene sfruttato il design pattern DAO, cioè degli oggetti di accesso ai dati che gestiscono l'accesso a questi ultimi, incapsulando le modalità di comunicazione con il DB.

Le Data Access Objects si fanno carico di gestire il codice SQL, mentre ciò è trasparente rispetto alle corrispondenti classi di dominio e di controllo.

A livello di logica dell'applicazione siamo fortemente orientati agli oggetti, infatti ragioniamo solo in termini di Domain Objects, cioè dei concetti pertinenti al dominio dell'applicazione, e non possiamo mai utilizzare i metodi di accesso diretto alla base dati forniti dai DAO.

Il design pattern DAO quindi ci dà un approccio diverso alle future modifiche e manutenzioni poiché i dati vengono gestiti da apposite classi, separando cosi in modo migliore le logiche a vantaggio di chi dovrà gestire il codice in futuro.

Nel nostro progetto è la classe Intermedio.java si fa carico di gestire il codice SQL, i dati vengono gestiti da apposite classi, separando in modo leggibile il codice.



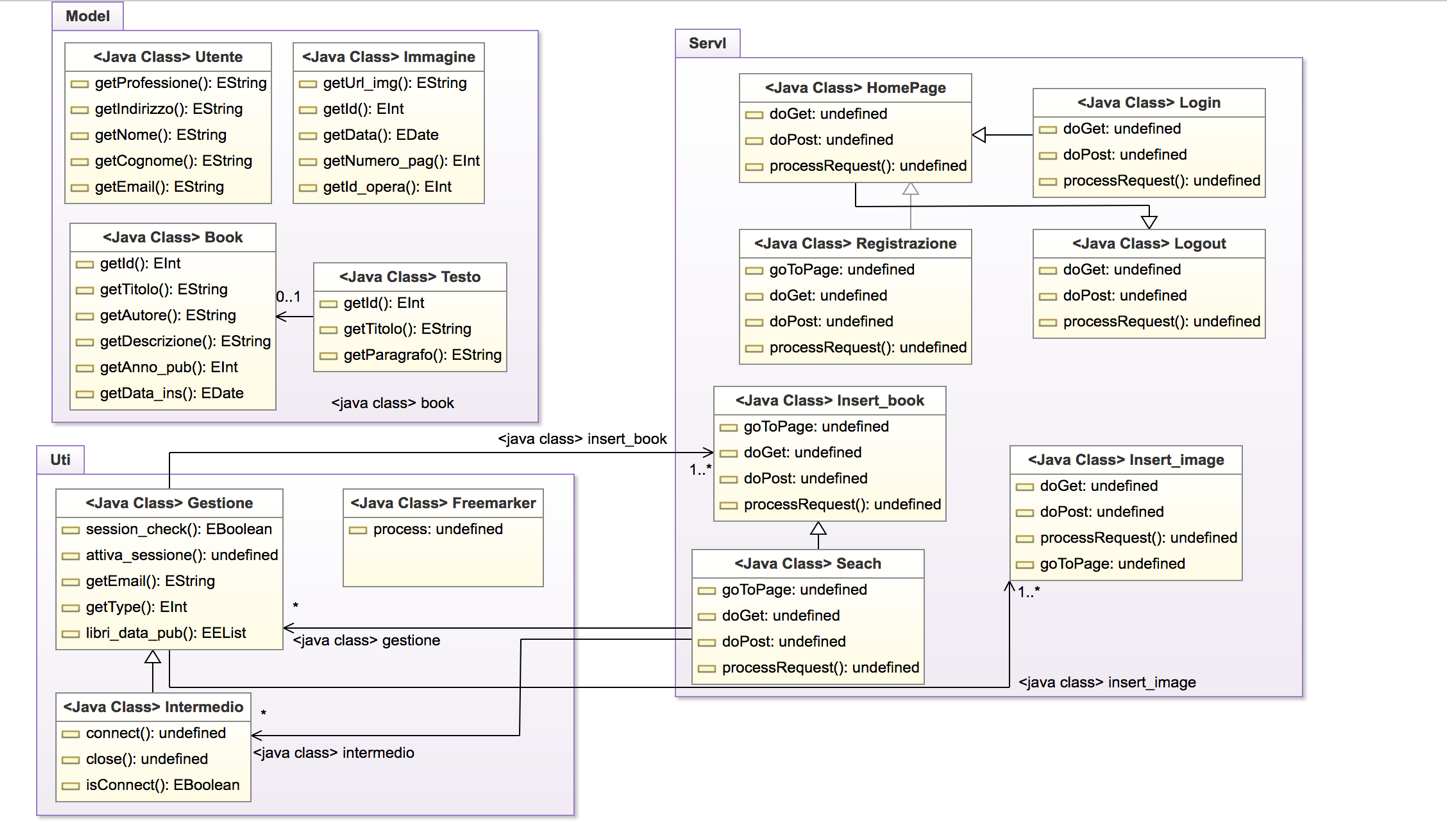
**Progetto OOP – Deliverables 2015-2016**

In generale, lo scopo principale di un Component Diagram è di mostrare le relazioni strutturali tra le componenti di un sistema (una componente è un elemento fisico rimpiazzabile del sistema). Le componenti coinvolte sono le seguenti:

* ***UserInterface***: Rappresenta le interfacce grafiche con le quali l'utente utilizza il sistema. Comunicherà con "Uso libreria" attraverso l’interfaccia “library” passandogli i valori per una eventuale ricerca.
* ***OperatorInterface***: Rappresenta le interfacce grafiche con le quali l’operatore utilizza il sistema.  Si Comunicherà con “Gestione utenza e libreria” attraverso l’interfaccia “Gestisce”. Operator interface è una view comune a Amministratore e Validatori, questo per accorpare le interfacce gestionali.
* ***Database***: Rappresenta la parte Model del sistema: il database JDBC, il quale si occuperà di contenere i dati.
* ***Gestione dati***: Costituisce la parte del sistema che si occupa di estrapolare i dati dalla base dati
* ***Gestione utenza e libreria***: La parte controller del sistema dove saranno accorpate tutte le procedure funzionali.
* ***Uso libreria***: Raggruppa tutte le funzionalità per utilizzare la libreria.
* ***Upload***: Rappresenta le funzionalità di upload di un’immagine.

**Progetto OOP – Deliverables 2015-2016**

**Class Diagram dell’implementazione del sistema**



Il class diagram consente di descrivere *tipi di entità*, con le loro caratteristiche e le eventuali relazioni fra questi tipi. Per il nostro sistema sono state individuate le seguenti classi:

* La classe **Utente** che racchiude i dati di tutti gli utenti e permette di fare operazioni su di essi.

**Progetto OOP – Deliverables 2015-2016**

**Design Decision**

Tra le scelte architetturali più importanti rientra quella di creare un’applicazione desktop o una applicazione web; uno dei vantaggi di un’applicazione desktop è sicuramente la possibilità di essere utilizzata in assenza di rete e di mantenere file in locale, quest’ ultima peculiarità porterebbe ad un minor sovraccarico del server.

Tuttavia, il problema di utilizzare i file in assenza di rete, può essere superato fornendo all’utente la possibilità di scaricare le informazioni da lui ricercate in un qualunque formato.

È ritenuta migliore la scelta architetturale di utilizzare una applicazione web per la sua facilità di utilizzo per gli tutti gli utenti e la possibilità di accedere al sistema da qualunque piattaforma.

Per la creazione del Sistema è stato scelto di utilizzare un’architettura web-oriented, che si presta alla gestione della multiutenza, il sistema infatti prevede che ci siano sette diversi tipi di utenti e che ognuno abbia diversi scopi.

Per la gestione della multiutenza si prevede che il database sia creato in modo da avere tre diverse tabelle che gestiscano:

* Utenti
* Gruppi di Utenti
* Servizi

**Progetto OOP – Deliverables 2015-2016**

Nella tabella utenti ci saranno i dati di ciascun utente, nella tabella gruppi di utenti ci saranno i sette diversi tipi di utenti e nella tabella servizi le funzionalità a cui possono accedere i gruppi di utenti.

Gli amministratori del sistema si occuperanno di assegnare un utente ad un particolare gruppo di utenti, ad eccezione per l’utente avanzato che viene assegnato al suo gruppo automaticamente dopo la registrazione.

L’applicazione web dovrà fornire a ciascun tipo di utente la sua interfaccia, in modo da permettergli di svolgere la sua mansione.