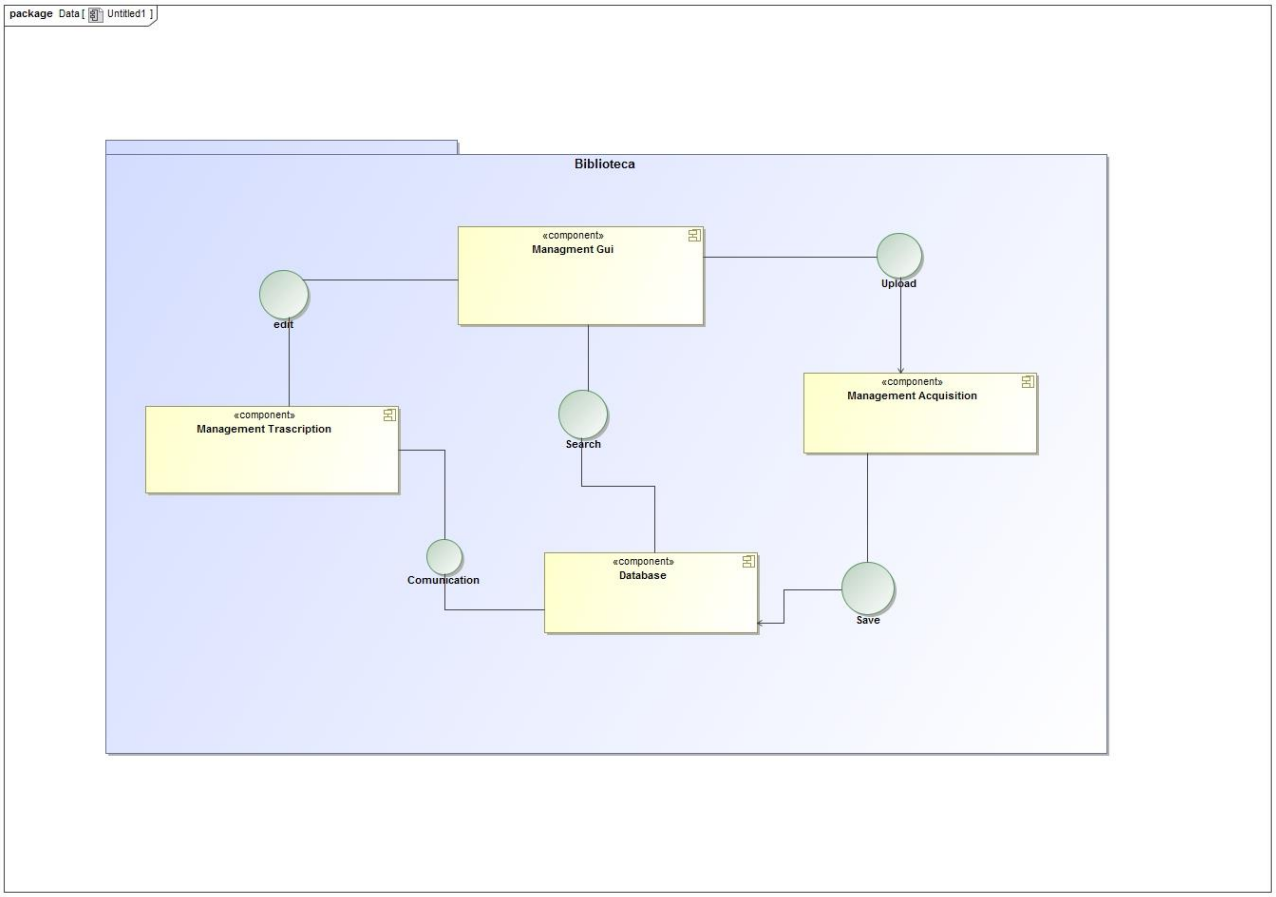
# Modello dell’architettura software




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:

**Management 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.

**Management Acquisition** : Rappresenta la componente che permette la scannerizzazione del documento. Si interfaccia con Management Gui attraverso l'interfaccia "Upload", con la quale l'utente acquisitore avvia la componente e successivamente si interfaccia con il Database attraverso "Save", passandogli l'immagine del libro la quale verrà salvata nel database.

**Management Trascription** : Rappresenta la componente che permette al Trascrittore di trascrivere il libro. Si interfaccia con il database attraverso "Comunication", per richiedere l'immagine di un libro. Inoltre si interfaccia con Management Gui attraverso "Edit", con il quale il Trascrittore invia la richiesta per avviare il text editor TEI.

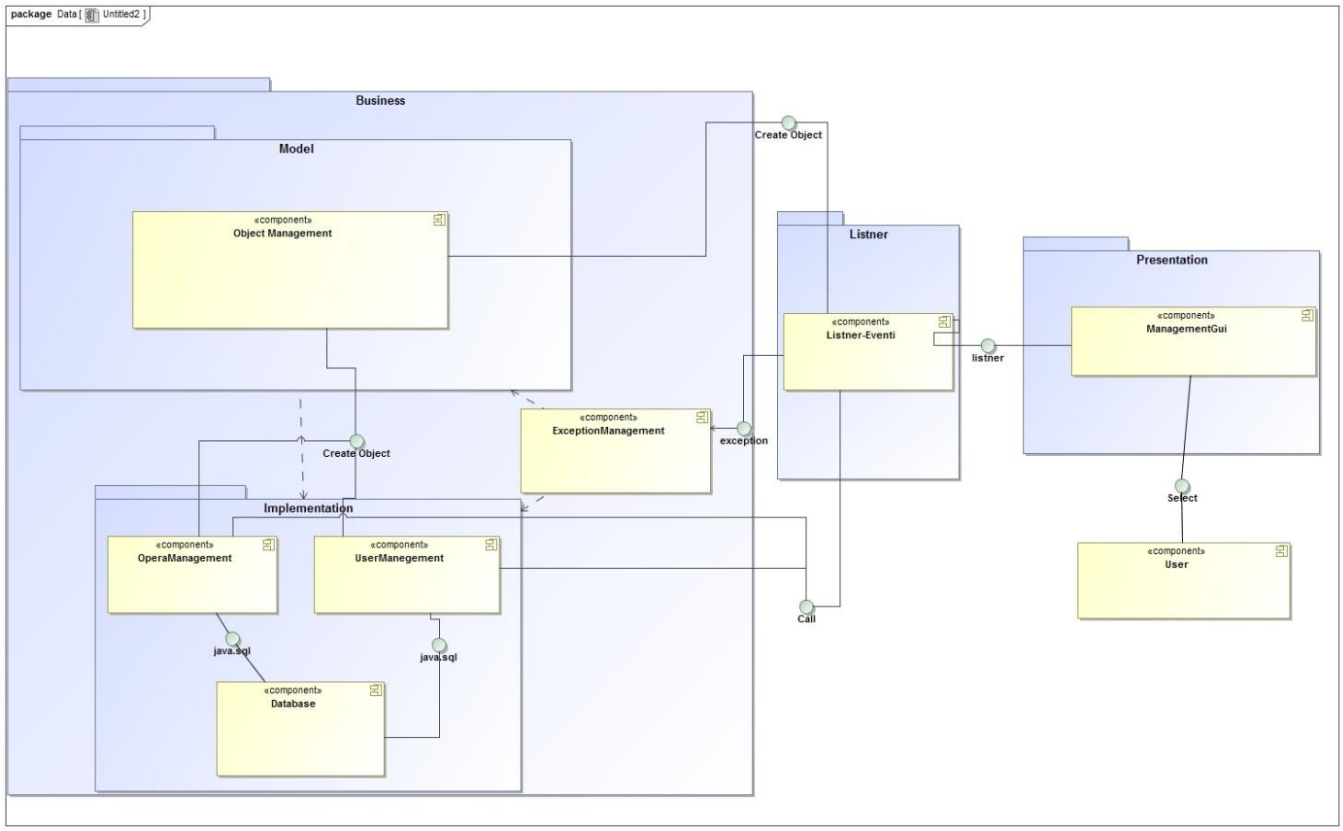
## ARCHITETTURA CLIENT-SERVER


Abbiamo scelto questa architettura in quanto i sistemi client/server sono un'evoluzione dei sistemi basati sulla condivisione semplice delle risorse (nel nostro caso libri). Il modello client/server è particolarmente consigliato per delle reti che necessitano un elevato livello di fiducia, e i suoi principali vantaggi sono:

- **Risorse centralizzate**: dato che il server è al centro della rete, può gestire delle risorse comuni a tutti gli utenti, come ad esempio un database centralizzato, per evitare i problemi di ripetizione e contraddizione.
- **Una sicurezza migliore**: dato che ci sono un numero limitato di utenti c'è maggiore sicurezza di dati.
- **Un'amministrazione a livello del server**: considerata la poca importanza dei client in questo modello, hanno meno bisogno di essere amministrati.
- **Una rete evolutiva**: grazie a questa architettura è possibile cancellare o aggiungere i client senza disturbare il funzionamento della rete e senza modifiche importanti.

# DESCRIZIONE DELL'ARCHITETTURA SCELTA

## (Client-Server)




**Chi sono i Client e cosa fanno:** Il software client "Presentation" in questo caso, è di limitata complessità, limitandosi normalmente ad operare come interfaccia verso il server. I client accedono ai servizi o alle risorse di un'altra componente, detta *server*, attraverso un'interfaccia (Gui) con la quale effettuano trascrizioni e/o acquisizioni delle immagini dei manoscritti oppure visualizzano opere e/o titoli in base al tipo di utente.

**Chi sono i Server e cosa fanno:** Il software *server "Business"*, oltre alla gestione logica del sistema, deve implementare tutte le tecniche di gestione degli accessi, allocazione e rilascio delle risorse, condivisione e sicurezza dei dati o delle risorse "OperaManagement" "UserManagement". Nel nostro caso attribuiamo a esso la gestione del Database. Inoltre si occupa anche della gestione degli eventi scaturiti dall'utente grazie alla componente ListenerEventi.

# DESCRIZIONE SCELTE E STRATEGIE ADOTTATE

## MVC

Il pattern architetturale utilizzato è il Model-View-Controller. Questo pattern è molto diffuso nello sviluppo di sistemi software, in particolare nell'ambito della programmazione orientata agli oggetti perché è in grado di separare la logica di presentazione dei dati dalla logica di business.

Esso è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

- Il **Model** fornisce i metodi 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 e li attua modificando lo stato degli altri due componenti.

Nella nostra applicazione questi ruoli sono ben distinti grazie all'utilizzo dei package. Nell'ordine:

- Il package *Business* si occupa della parte *Model*, implementando quindi la logica di business che riguarda l'accesso, la modifica e il recupero dei dati dell'applicazione.
- Il package *Presentation* svolge il ruolo di *View*, quindi è delegato alla gestione dell'interfaccia utente.
- Il package *Listener* è il nostro *Controller*, il quale riceve i comandi dell'utente tramite il View (Presentation) ed effettua le operazioni richieste. Insieme al package Model si occupa quindi di implementare tutta la logica di business del sistema.

## SINGLETON

Abbiamo deciso di utilizzare il design pattern creazionale *Singleton* per la gestione della connessione al database nella nostra applicazione.

Esso è uno dei pattern fondamentali della programmazione ad oggetti. Lo scopo di questo pattern è di garantire che di una determinata classe venga creata una e una sola istanza, e di fornire un punto di accesso globale a tale istanza.

Nel nostro caso specifico abbiamo avuto la necessità di modificare leggermente l'utilizzo originario di questo pattern; infatti a differenza dell'implementazione standard che prevede un metodo statico *"get"* per la creazione e la restituzione di un'istanza della classe, abbiamo deciso di utilizzare il metodo *"get"* per l'inizializzazione e la restituzione di un attributo della classe (l'oggetto Connection dichiarato *static* e *final*), in quanto per effettuare operazioni nel database abbiamo bisogno solamente di un oggetto Connection, piuttosto che dell'istanza della classe.

## JAVA

Per la costruzione del sistema useremo la **piattaforma software** Java, in quanto, codesta è stata testata, perfezionata, estesa e provata da una community dedicata di sviluppatori, architetti e utenti appassionati di Java. Java è stato progettato per consentire lo sviluppo di applicazioni per sistemi portatili con prestazioni elevate per la più ampia gamma di piattaforme di elaborazione possibile. Rendendo disponibili le applicazioni in ambienti eterogenei, le aziende sono in grado di fornire più servizi, aumentare la produttività, la comunicazione e la collaborazione degli utenti finali e di ridurre significativamente il costo della proprietà delle applicazioni aziendali e di quelle di consumo.

## SQLITE

Per quanto riguarda il database useremo **Sqlite.** Si tratta di una database library concepita in linguaggio C con lo scopo di implementare un DBMS SQL da utilizzare all'interno di un'applicazione. SQLite è un software Open Source che può essere liberamente utilizzato e modificato. L'uso di **SQLite** presenta numerosi vantaggi; innanzitutto si tratta di una libreria che richiede pochissimo spazio nel disco rigido, all'incirca 250 KB e questa caratteristica contribuisce a rendere i processi ad essa correlati molto rapidi e poco dispendiosi in termini di risorse.

Un'altra caratteristica da segnalare è quella riferita all'aderenza delle transazioni di **SQLite** allo schema **ACID** (**A**tomicity, **C**onsistency, **I**solation and **D**urability), cioè:

1. **Atomicità**, per cui ogni transazione è indivisibile durante l'esecuzione;
2. **Consistenza**, per cui viene salvaguardata la consistenza del database all'inizio, durante e alla fine della transizione;
3. **Isolamento**, per cui ogni transazione viene eseguita isolatamente e indipendentemente dalle altre;
4. **Durabilità**, per cui le modifiche apportate divengono persistenti e non vengono perdute;

## JAVA SWING

Per lavorare l'interfaccia grafica useremo SWING. Il package Swing, è composto da componenti realizzati completamente in Java, ricorrendo unicamente alle primitive di disegno più semplici, tipo "traccia una linea" o "disegna un cerchio", accessibili attraverso i metodi dell'oggetto Graphics, un oggetto AWT utilizzato dai componenti Swing per interfacciarsi con la piattaforma ospite. Le primitive di disegno sono le stesse su tutti i sistemi grafici, e il loro utilizzo non presenta sorprese: il codice java che disegna un pulsante Swing sullo schermo di un PC produrrà lo stesso identico risultato su un Mac o su un sistema Linux. Questa architettura risolve alla radice i problemi di uniformità visuale, visto che la stessa identica libreria viene ora utilizzata, senza alcuna modifica, su qualunque JVM. Liberi dal vincolo del "minimo comune denominatore", i progettisti di Swing hanno scelto di percorrere la via opposta, creando un package ricco di componenti e funzionalità spesso non presenti nella piattaforma ospite.