PROGETTO: fumetteria ****

Data Creazione: 24/08/2015

Data ultima modifica: 06/09/2015

**Documento di Progetto**

**Versione 1.0**

**Elisa Antolli**

**Alice Culaon**

**Diego Pillon**

INDICE

1. MODELLO TABELLA DELLE REVISIONI 3

2. SCOPO DEL DOCUMENTO 3

3. AUTORI DEL DOCUMENTO 3

4. GLOSSARIO 3

5. Definizione dell’ architettura del sistema 4

6. Le tecnologie 11

7. Moduli del sistema 12

8. Dinamica del sistema 14

9. Diagramma delle classi 14

10. Dizionario dei dati 15

11. prototipi 15

1. MODELLO TABELLA DELLE REVISIONI

In questo modello viene

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev./Ver.** | **Data** | **Descrizione** | **Autore** |
| 1/1.0 | 24-08-2015 | Creazione del documento | Elisa Antolli |
| 2/1.0 | 03-09-2015 | Add. Tecnologie | Elisa Antolli |
| 3/1.0 | 06-09-2015 |  |  |
|  |  |  |  |
|  |  |  |  |
| **Tot. Rev. 1** |  | **Versione corrente 1.0** |  |

1. SCOPO DEL DOCUMENTO

Lo scopo di questo documento è descrivere e definire il sistema in modo più specifico degli altri documenti già presentati; il documento è rivolto al team di sviluppo per servire di base all’ implementazione del sistema.

1. AUTORI DEL DOCUMENTO
2. Antolli Elisa;
3. Culaon Alice;
4. Pillon Diego;
5. GLOSSARIO

I termini del Glossario vanno ordinati in modo crescente alfabeticamente.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Termine** | **Descrizione** | | | **Paragrafo** | |
|  |  | | |  | |
|  | |  |  | |  |
| **Termini del Glossario:** | | | | | |

1. Definizione dell’ architettura del sistema

In questa sezione l’architettura del sistema viene spiegata in più dettaglio, specificando le tecnologie utilizzate - come vengono utilizzate e dove vengono utilizzate.

* 1. **L’ architettura di base**

Il sistema viene sviluppato in tecnologia web - un’applicazione web indica, in generale, i sistemi informatici progettati per l'utilizzo tramite un browser su internet, applicazioni sviluppate utilizzando tecnologie web HTML, JavaScript e CSS. Può essere eseguito da un server HTTP (*Host web*) o localmente sul dispositivo dell'utente. La funzione di *server web* è quello di ricevere una richiesta (richiesta) e ritorno (risposta) al *browser* dell’utente. O *browser* consente all'utente di richiedere una risorse, e quando il server risponde a una richiesta si trovano caratteristiche come: pagine HTML, immagini e documenti PDF che vengono poi visualizzati per l'utente. Di solito i server inviano istruzioni per il *browser* scritta in HTML. HTML dice al *browser* come presentare i contenuti per l'utente web. Il sistema sarà depositato su un web server progettato per servire le richieste di ogni cliente simultaneamente. In generale i dati vengono salvati ed estratti con il supporto di un Database Server e per la costruzione delle pagine dinamiche, viene utilizzato un’applicazione di supporto basata in *Servlets*.

Per questo abbiamo scelto utilizzare il modello di architettura MVC – *Model View Controller;* Il MVC è stato creato negli anni '70, ed è ancora un modello applicabile in varie applicazioni, in particolare le applicazioni web. Con la crescente complessità delle applicazioni sviluppate, si diventa essenziale la separazione tra i dati (*Model*) e il layout (*View*). Pertanto, le modifiche apportate al layout non influenza la manipolazione dei dati, e possono essere riorganizzati senza modificare il layout.

* Il *Model* è la parte dove vengono specificate le rappresentazione di “dominio” – cioè, quelle che dicono come viene fatta la struttura dei dati dentro del sistema – ad esempio una classe “Author” con i suoi determinati attributi e metodi *get/set*. In poche parole sono la rappresentazione degli elementi del dominio e di interazione con gli strumenti di persistenza.
* Il *View* è la parte dove il *Model* viene dimostrato all’ utente, in una determinata forma permettendo la interazione tra utente ed il sistema. In modi tecnici, sono le schermate con risposta HTML (JSP,ASP, PHP).
* La *Controller* sono i processi e risposte agli eventi, tipicamente azioni degli utenti che possono venire a cambiare il Model. Nella controller viene fatta la validazione dei dati ed è anche il luogo dove i valori inseriti dagli utenti vengono ‘filtrati’.

Una semplice dimostrazione grafica del modello MVC viene riportata sotto:

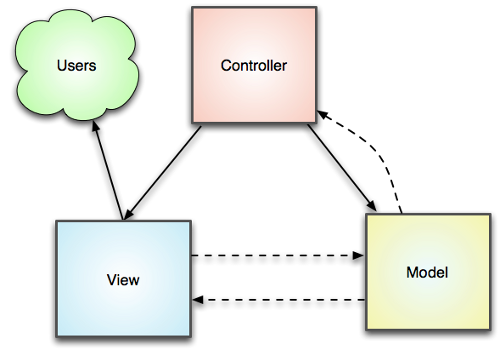


Figura 1 Rappresentazione generica MVC

Per lo sviluppo di questa architettura, abbiamo utilizzato un *framework* di supporto chiamato *Spring Framework* - un *framework* open source che dà la possibilità di creare e gestire un progetto diviso in una struttura MVC, facilitando lo sviluppo attraverso della fornitura di innumerevole *library* e risorse che saranno spiegate in più dettaglio nella prossima sezione.

* 1. **La struttura Spring Framework**

Il sistema è diviso in tale forma a seguire lo standard del Spring Framework. L'adozione di questo standard permette allo sviluppatore di concentrarsi sull'attuazione delle caratteristiche specifiche dell’applicazione, delegando al Framework, come lo Spring, il compito di specificare la dipendenza di alcuni oggetti. La struttura dello Spring è presentata nella figura sotto:

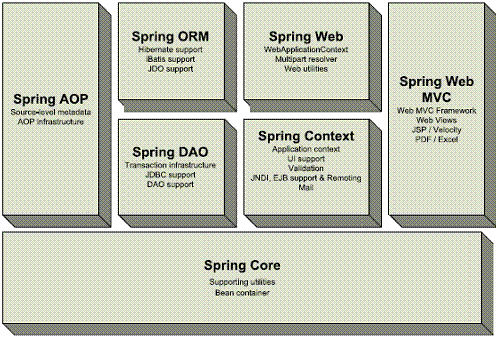


Figura 2 Struttura Framework

Il modulo Spring Core rappresenta le caratteristiche principali dello Spring, in cui l'elemento principale è il *BeanFactory* - si tratta di una implementazione dello pattern *Factory*, incaricato di applicare *l’Inversion of Control* che consente l’accoppiamento tra la configurazione e la specificazione delle dipendenze e la loro logica di programmazione.  
  
Il modulo *Spring Context* dello Spring si trova sopra il pacchetto principale e fornisce un modo per accedere il *beans* con uno stile di struttura. Il *Spring Context* fornisce il supporto per i servizi aziendali, come *Java* *Naming* and Directory Interface ™ (JNDI), Enterprise JavaBeans (EJB), e-mail, convalidazione, l'internazionalizzazione, la propagazione di eventi, carico di risorse e la creazione trasparente di contesti.

Il modulo Spring AOP è il modulo che permette l’implementazione di programmazione orientata a aspetti, che in questo progetto non usiamo, quindi non viene approfondita.

Il modulo Spring DAO fornisce un livello di astrazione per JDBC, eliminando gran parte del codice necessario per interagire con un database. Inoltre fornisce un modo semplice per gestire la manipolazione delle eccezioni e codici di errore emessi da diversi fornitori di database. Anche fornisce un modo per eseguire la gestione di transazioni programmatiche e dichiarative, non solo per le classi che implementano le interfacce speciali, ma per tutti gli oggetti Java (POJO). Il progetto Spring Framework consente l'integrazione diretta in alcuni dei popolari Api’s ORM, come JDO, Hibernate e iBatis.

Il modulo ORM, tuttavia, prevede l'integrazione dello Spring con altri framework di persistenza di oggetti, come Hibernate e iBatis. Gli strumenti ORM consentono agli sviluppatori di realizzare il principio fondamentale di progettazione orientata agli oggetti: l'incapsulamento. Questo consente a un client di interagire con un oggetto senza conoscenza dei loro dettagli di implementazione.  
  
Per fornire funzionalità specifiche per progetti Web, esiste il modulo Spring Web. Sono fornite caratteristiche come i componenti per il caricamento dei file e il supporto per l'utilizzo di *Inversion* *of Control*. Esso fornisce anche altre caratteristiche, come l'inizializzazione dei contesti utilizzando *listeners* di *servlet* ed un contesto di applicazioni orientata per la *Web*.

Il modulo Spring Web MVC fornisce un'implementazione della Model-View-Controller per le applicazioni Web. L’implementazione MVC non è solo una implementazione comune - fornisce una separazione pulita tra il codice del modello, del dominio e dei moduli web. Consente anche che tutte le altre caratteristiche dello Spring Framework come validazione, vengono utilizzate.

* 1. **La divisione del sistema**

Il sistema è suddiviso in pacchetti che seguono la struttura MVC dello Spring Framework:

* **com.progetto.fumetteria.configuration**

Qua sono realizzate le configurazioni base del sistema.

Nella classe AppConfig.java sono fatte gli “*import”* dei file xml di impostazioni, ad esempio le configurazione *Hibernate*. In questa classe sono fatte anche le configurazioni del tipo di *encoding*, luogo dei file (cartelle) ed anche i tipi di *views* che sono utilizzate.

Nella classe Initializer.java sono fatte alcune impostazione di inizializzazioni della applicazione, ad esempio l’impostazione della ruota iniziale della applicazione. Vengono fatte anche le impostazione dei tipi di “*annotation*” che il progetto utilizza ed anche quali sono i *listeners* del sistema.

* **com.progetto.fumetteria.controller**

Il pacchetto controller è responsabile per manipolare il cambiamento di informazione tra la *view* ed i servizi che sono a disponibilità della applicazione. La controller è anche responsabile per il *mapping* delle ruote ed il “reindirizzamento” degli utenti alle pagine richieste.

Le classi che appartengono a questo pacchetto sono:

* AdminController - Questa è la controller che crea la ruota della pagina principale che contengono tutte le ruote per le funzione disponibili solamente all’amministratore del sistema.
* LoginController - Questa controller contiene le valutazione degli utenti del sistema; tanto un utente finale, quanto un utente amministratore.
* AuthorController - Questa controller è responsabile per le azioni d’intermediazione tra le *view* ed il *model* della gestione di autori.
* ComicController – Questa controller è responsabile per le azioni d’intermediazione tra le *view* ed il *model* della gestione di comic (compreso anche il requisito *RFBO09.Gestione Stock*).
* ErrorController – Questa controller dovrebbe gestire tutte le *exception* e fare un *redirect* ad una pagina Default di errori.
* GenreController - Questa controller è responsabile per le azioni d’intermediazione tra le *view* ed il *model* della gestione di generi.
* HomeController – Questa controller gestisce le funzionalità e ruote della pagina principale [con lista (*menubar)* dei generi, lista dei fumetti, filtraggi fumetti, ecc.] del percorso *Front* *Office*.
* NoteController - Questa controller è responsabile per le azioni d’intermediazione tra le *view* ed il *model* della gestione di note (commentari).
* NoticeController - Questa controller è responsabile per le azioni d’intermediazione tra la *view* ed il *model* della creazione di avviso di disponibilità dei fumetti.
* PublishingHouseController - Questa controller è responsabile per le azioni d’intermediazione tra le *view* ed il *model* della gestione di case editrice.
* ReserveController – Questa controller è responsabile per le azioni d’intermediazione tra le *view* ed il *model* della gestione di prenotazioni.
* SuggestionController - Questa controller è responsabile per le azioni d’intermediazione tra le *view* ed il *model* della gestione dei suggerimenti.
* UserController - Questa controller è responsabile per le azioni d’intermediazione tra la *view* ed il *model* della gestione di tutti tipi di utenti.
* ComicFrontController – Questa controller gestisce le funzionalità e ruote della pagina intermediaria “Visualizzare un singolo Fumetto” del percorso *Front* *Office*.
* GenreFrontController - Questa controller gestisce le funzionalità e ruote della pagina intermediaria di visualizzazione dei fumetti di determinato genero del percorso *Front* *Office.*
* ReserveFrontController – Questa controller gestisce le funzionalità e ruote di creazione di una prenotazione.
* SuggestionFrontController – Questa controller gestisce le funzionalità e ruote di creazione di un suggerimento.
* UserFrontController - Questa controller è responsabile per le azioni d’intermediazione tra le *view* ed il *model* della gestione di utenti clienti (utenti non amministratori).
* **com.progetto.fumetteria.dao**

Il DAO è responsabile per la comunicazioni tra l’applicazione e la base dei dati. E’ dove sono fatte le connessione con il data base, inserimenti, cancellamenti, ecc. Queste operazioni sono fatte utilizzando un *framework* ORM denominato *Hibernate* – questo *framework* ha l’obbiettivo di facilitare l’interazione tra l’applicazione e la base dei dati, in maniera tale da tornare questo lavoro più trasparente al programmatore.

Le classi che appartengono al DAO sono:

* GenericDao: Questa è la classe più importante del pacchetto DAO, perché questa è classe astratta che contiene tutta la logica per realizzare le chiamate all’operazione del database: alcuni diversi tipi di filtraggi, inserimenti, cancellamenti ed aggiornamenti. Tutte le altre classi sono responsabile appena per estendere questa classe e passare il tipo di oggetto che se deve lavorare sopra.
* AuthorDao: Questa classe è la responsabile per il tipo Autore.
* ComicAuthorDao: Questa classe è la responsabile per il tipo “relazione tra Autori e Fumetti”.
* ComicDao: Questa classe è la responsabile per il tipo Fumetto.
* GenreDao: Questa classe è la responsabile per il tipo Genere.
* NoteDao: Questa classe è la responsabile per il tipo Note (commentari).
* PublishingHouseDao: Questa classe è la responsabile per il tipo Casa Editrice.
* UserDao: Questa classe è la responsabile per il tipo Utente.
* ReserveDao: Questa classe è la responsabile per il tipo Prenotazione.
* ComicReserveDao: Questa classe è la responsabile per il tipo “relazione tra Fumetto e Prenotazione”.
* NoticeDao: Questa classe è la responsabile per il tipo Notificazione.
* UserRoleDao: Questa classe è la responsabile per il tipo Ruolo dell’utente.
* PhoneDao: Questa classe è la responsabile per il tipo Telefono.
* **com.progetto.fumetteria.model**

Le classi “model” rappresentano un *mapping* oggetto-relazione della base dei dati del sistema. È utilizzato per il passaggio di informazioni tra i livelli del modello MVC (*Model-View-Controller*) ed anche è utilizzato per il *framework* *Hibernate* per fare il *mapping* della base dei dati.

Le classi che appartengono alla “model” sono:

* Author: Questa classe contiene tutte le informazioni dell’autore – ad esempio, la tabella “*author*” della base dei dati contiene la colonna “*name*”, allora questa classe avrà un attributo “*name*” con i suoi rispettivi *get/set*.
* ComicAuthor: Questa classe contiene tutte le informazioni di quello che sarebbe la relazione tra le due classi “Author” e “Comic” – ad esempio, la tabella “*author*” è in relazione con la tabella “*comic*” della base dei dati, e loro contengono una classe intermediaria che gestisce il loro rapporto, allora, la classe *ComicAuthor* serve per fare il mapping di questa tabella intermediaria, come già spiegato nell’esempio precedente.
* Comic: Questa classe contiene tutte le informazioni del fumeto – ad esempio, la tabella “*comic*” della base dei dati contiene la colonna “*title*”, allora questa classe avrà un attributo “*title*” con i suoi rispettivi *get/set*.
* Genre: Questa classe contiene tutte le informazioni del genere – ad esempio, la tabella “*genre*” della base dei dati contiene la colonna “*name*”, allora questa classe avrà un attributo “*name*” con i suoi rispettivi *get/set*.
* Note: Questa classe contiene tutte le informazioni del commentario (nota) – ad esempio, la tabella “*note*” della base dei dati contiene la colona “*status*”, allora questa classe avrà un attributo “*status*” con i suoi rispettivi *get/set*.
* PublishingHouse: Questa classe contiene tutte le informazioni della casa editrice – ad esempio, la tabella “*publishing\_house*” della base dei dati contiene la colonna “*name*”, allora questa classe avrà un attributo “*name*” con i suoi rispettivi *get/set*.
* User: Questa classe contiene tutte le informazioni dell’utente – ad esempio, la tabella “*user*” della base dei dati contiene la colonna “*email*”, allora questa classe avrà un attributo “*email*” con i suoi rispettivi *get/set*.
* Reserve: Questa classe contiene tutte le informazioni della prenotazione – ad esempio, la tabella “*reserve*” della base dei dati contiene la colonna “*total*”, allora questa classe avrà un attributo “*total*” con i suoi rispettivi *get/set*.
* ComicReserve: Questa classe contiene tutte le informazioni di quello che sarebbe la relazione tra le due classi “Comic” e “Reserve”, lavorando nella stessa maniera della classe “ComicAuthor”.
* Notice: Questa classe contiene tutte le informazioni dell’avviso – ad esempio, la tabella “*notice*” della base dei dati contiene la colonna “*email*”, allora questa classe avrà un attributo “*email*” con i suoi rispettivi *get/set*.
* Phone: Questa classe contiene tutte le informazioni di un telefono che un utente può avere – ad esempio, la tabella “*phone*” della base dei dati contiene la colonna “*id\_user*” che serve per dire che il telefono “*xxx xxxxxx*” ha come id l’utente “*x*”, dove un utente può avere diversi numeri di telefono. Allora, questa classe avrà un attributo “*idUser*” con i suoi rispettivi *get/set*.
* **com.progetto.fumetteria.service**

Nel pacchetto service ci sono tutti i servizi in cui il DAO ci dà la possibilità di usufruire. Questo modulo del sistema è indipendente della programmazione realizzata per effettuare la connessione con la base dei dati. La sua utilità principale è “modularizzare” in una maniera più indipendente i livelli del modello MVC – ad esempio, se la forma come la comunicazione con la base dei dati fosse cambiata, i livelli “Controller” e “View” non sarebbero affettati. (Considerando che il cambiamento fatto utilizza lo stesso ‘*standard’* proposto);

Le classi che appartengono al pacchetto service sono:

* AuthorService: Questa classe è la responsabile per i servizi offerti del tipo autore.
* ComicAuthorService: Questa classe è la responsabile per i servizi offerti della relazione tra i tipi autori – fumetti.
* ComicService: Questa classe è la responsabile per i servizi offerti del tipo fumetto.
* GenericService: Questa classe è una classe astratta con tutte le operazioni che vengono ad affettare la base dei dati. E’ scritta in una maniera più comprensibile al programmatore e utilizzando gli oggetti e suoi rapporti con altri oggetti piuttosto del linguaggio del *database*. Le altre classi devono appena estendere questa e passare come parametro il suo tipo di oggetto.
* GenreService: Questa classe è la responsabile per i servizi offerti del tipo genere.
* NoteService: Questa classe è la responsabile per i servizi offerti del tipo commentario (nota).
* PublishingHouseService: Questa classe è la responsabile per i servizi offerti del tipo casa editrice.
* UserService: Questa classe è la responsabile per i servizi offerti del tipo utente.
* ReserveService: Questa classe è la responsabile per i servizi offerti del tipo prenotazione.
* ComicReserveService: Questa classe è la responsabile per i servizi offerti della relazione tra i tipi prenotazioni-fumetti
* NoticeService: Questa classe è la responsabile per i servizi offerti del tipo avviso.
* PhoneService: Questa classe è la responsabile per i servizi offerti del tipo telefono.
* **com.progetto.fumetteria.util**

In questo pacchetto ci sono alcune classi per l’ausilio alla programmazione, ad esempio la edizione dei formati numerici, le classi con valori statici ecc.

* **src.main.webapp.WEB-INF.jsp**

Questo è il pacchetto dove ci sono le schermate del sistema: suddiviso in *backoffice* e *frontoffice*.

1. Le tecnologie   
   1. **Le tecnologie web – client side**

Le tecnologie di base sono le tecnologie principale ed essenziale per creare un sistema web.

* HTML – *l’HTML* è un abbreviazione per l’espressione inglese *Hyper Text Markup Language* che significa “Linguaggio di formattazione di ipertesto”. E’ un linguaggio utilizzato per produrre pagine web perché documenti scritti in html possono essere interpretati per i browser.
* CSS – *Cascading Style Sheets* oppure CSS è un linguaggio di fogli di stile utilizzato per definire la presentazione di documenti scritti in un linguaggio di formattazione come l’HTML. Il suo principale beneficio è promuovere la separazione tra il formato e il contenuto di un documento: invece di mettere la formattazione nel documento, lo sviluppatore crea un link che punta alla pagina che contiene lo stile. Quindi quando si vuole modificare layout del sito, basta modificare il file del link. Il progetto è basato anche su tecnologia CSS3 che contiene delle risorse in più delle precedenti. *Mozilla Firefox* e *Google Chrome*, sono i browser che hanno maggiore supporto per questa tecnologia, infatti i test vengono eseguiti su questi browser.
* JavaScript – *Javascript* è un linguaggio di programmazione *client side.* E’ utilizzato per controllare l’*HTML* ed il *CSS* per manipolare i comportamenti nella pagina senza necessariamente passare per il *server*, controllando il *browser*, realizzando comunicazione asincrona e cambiando il contenuto del documento. E’ stato concepito per essere un linguaggio di *script* orientato agli oggetti basato in prototipi, debolmente tipato e dinamico e funzioni di prima classe.
* jQuery – *Jquery* è una *library javascript cross-browser* sviluppata per semplificare gli *script client-side* che interagiscono con l’*HTML*. E’ *opensource* e la sintassi è stata sviluppata per semplificare la navigazione del documento *HTML*, la selezione i elementi *DOM* (Document object model – rappresentazione e interazione con oggetti in documenti di formattazione), creare animazioni, manipolare eventi e sviluppare applicazioni *AJAX*; questa *library* semplifica lo sviluppo di applicazioni *web* dinamiche di grande complessità.
* Bootstrap – *Bootstrap* è una raccolta di strumenti [liberi](https://it.wikipedia.org/wiki/Software_libero) per la creazione di [siti](https://it.wikipedia.org/wiki/Sito_web) e [applicazioni](https://it.wikipedia.org/wiki/Applicazione_web) per il [*Web*](https://it.wikipedia.org/wiki/Web). Questa contiene alcuni [modelli](https://it.wikipedia.org/wiki/Template) di progettazione basati su [*HTML*](https://it.wikipedia.org/wiki/HTML)e[*CSS*](https://it.wikipedia.org/wiki/CSS)*,* sia per la [tipografia](https://it.wikipedia.org/wiki/Tipografia), che per componenti dell'interfaccia, come [moduli](https://it.wikipedia.org/wiki/Form), bottoni e navigazione, e altri componenti dell'interfaccia, così come alcune estensioni opzionali di [*JavaScript*](https://it.wikipedia.org/wiki/JavaScript)*.*
* Ajax – *Ajax* è una metodologia che utilizza altre tecnologie come ad ed esempio *XML* e *JavaScript* per fare richieste asincrone al server e con le informazioni ritornate dal server cambia la pagina già caricata utilizzando *DOM* per cambiarla senza ricaricare nuovamente tutto il contenuto. Questa pratica riduce il traffico e migliora l’esperienza dell’utente.
  1. **Le tecnologie base – server side**
* Java – *Java* è un linguaggio di programmazione orientato agli oggetti. A differenza dei linguaggi convenzionali che sono compilati in codice nativo, il linguaggio *Java* viene compilato in un *bytecode* che viene eseguito in una macchina virtuale.
* Servlet - *Servlets* sono classi Java usate per estendere le funzionalità di un Server, sviluppate secondo una struttura ben definita che, quando installate e configurate su un *Server* che abbia l’implementazione di un *Servlet* *Container* è in grado di gestire le richieste ricevute dai *client Web*, come ad esempio i *browsers*.
* MySql – *MySql* è un sistema per gestire le basi dei dati (SGBD) che utilizza il linguaggio *SQL* (*Structured Query Language*) come interfaccia. E’ un sistema *open source*.
* JSP - *JavaServer Pages* (JSP) è una tecnologia che aiuta gli sviluppatori a creare pagine web dinamiche basate su HTML, XML o altri tipi di documenti. Per eseguire *JavaServer Pages*, un *web server* compatibile con un *Servlet Container* è necessario, ad esempio *Apache Tomcat*.
* Apache Tomcat - *Tomcat* è un *web server Java*, più precisamente, un *servlet container*. *Tomcat* fa l’implementazione delle tecnologie *Java Servlet* e *JavaServer Pages* (JSP); come *web server*, fornisce un server Web HTTP puramente in Java.
* Hibernate - *Hibernate* è un *framework* per il *mapping object-relational*, scritto nel linguaggio *Java*. Il *Hibernate* facilita la mappatura degli attributi tra una base tradizionale di dati ed il modello degli oggetti di un’applicazione, utilizzando file (XML) o *annotation* *Java*.

1. Moduli del sistema
   1. **Diagramma di contesto**

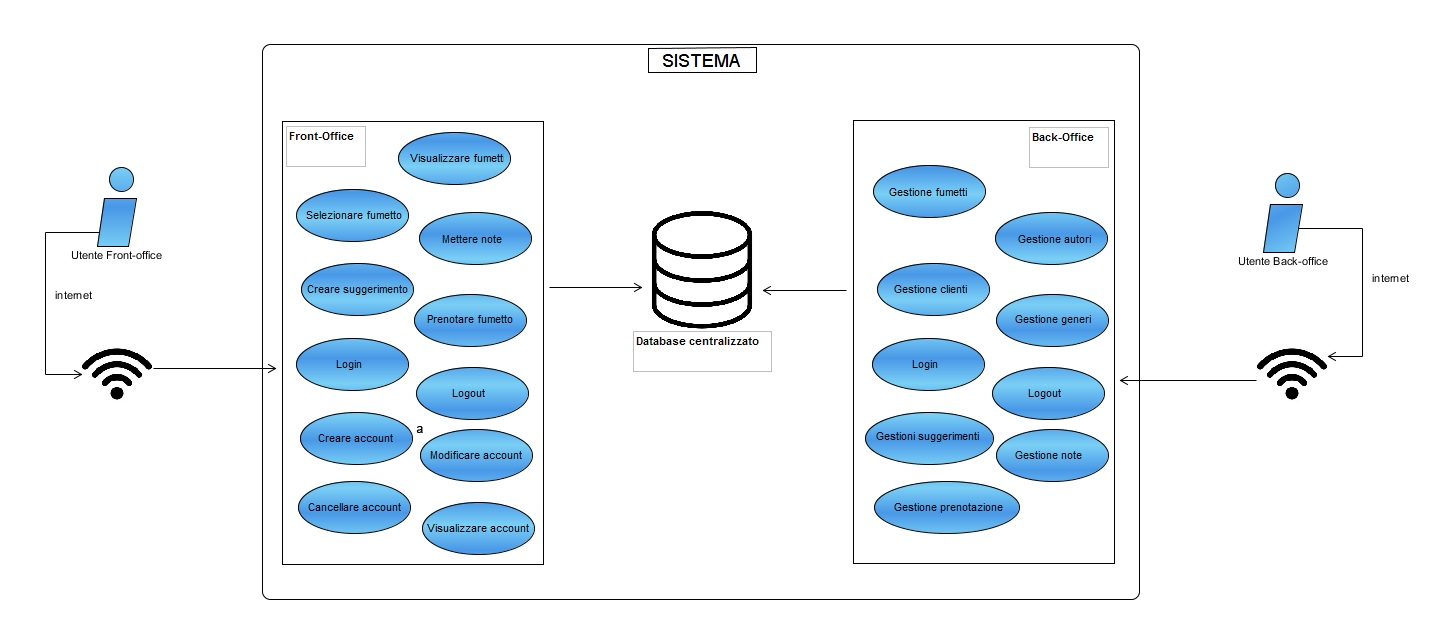


Figura 3 - Diagramma di contesto

* 1. **Front-Office**

Il modulo front-office del sistema ha lo scopo di raggruppare le operazione che sono dell’interesse dell’utente finale (del tipo ‘*customer’*), ad esempio fare la presentazione dei fumetti registrati nel sistema oppure eseguire una prenotazione di un determinato fumetto; La lista di tutte le possibili operazione del modulo front-office si incontra nel documento “Analisi dei Requisiti 4.4”.

* 1. **Back-Office**

Il modulo back-office del sistema ha lo scopo di raggruppare le operazione che un utente del tipo ‘amministratore’ può eseguire, ad esempio fare l’inserimento di un nuovo fumetto oppure cambiare lo *status* di una prenotazione. Il back-office del sistema funziona come la parte amministrativa del negozio, cioè, tutto ciò che serve per gestire il negozio.

* 1. **Database centralizzato**

Il database si occupa di salvare le informazioni utili al sistema; è centralizzato, e per quello garantisce che sia possibile gestire il sistema da diversi computer avendo sempre l’accesso agli stessi dati.

* 1. **Diagramma della suddivisione del progetto**

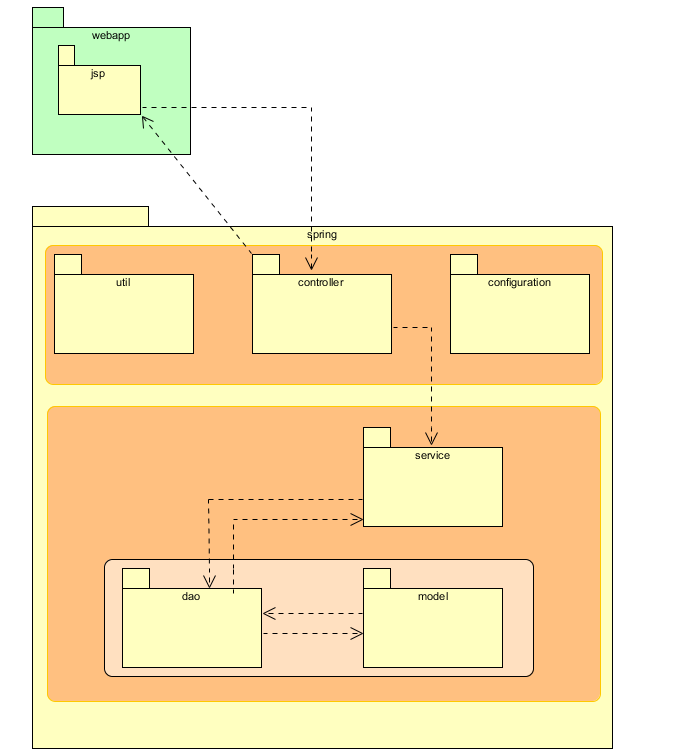


Figura 4 Suddivisione del progetto

1. Dinamica del sistema
   1. **Flussi principali**Diagramma di sequenza dei flussi principale – front-office e back-office
2. Diagramma delle classi

Diagramma entita’ relazione (DER) e ER

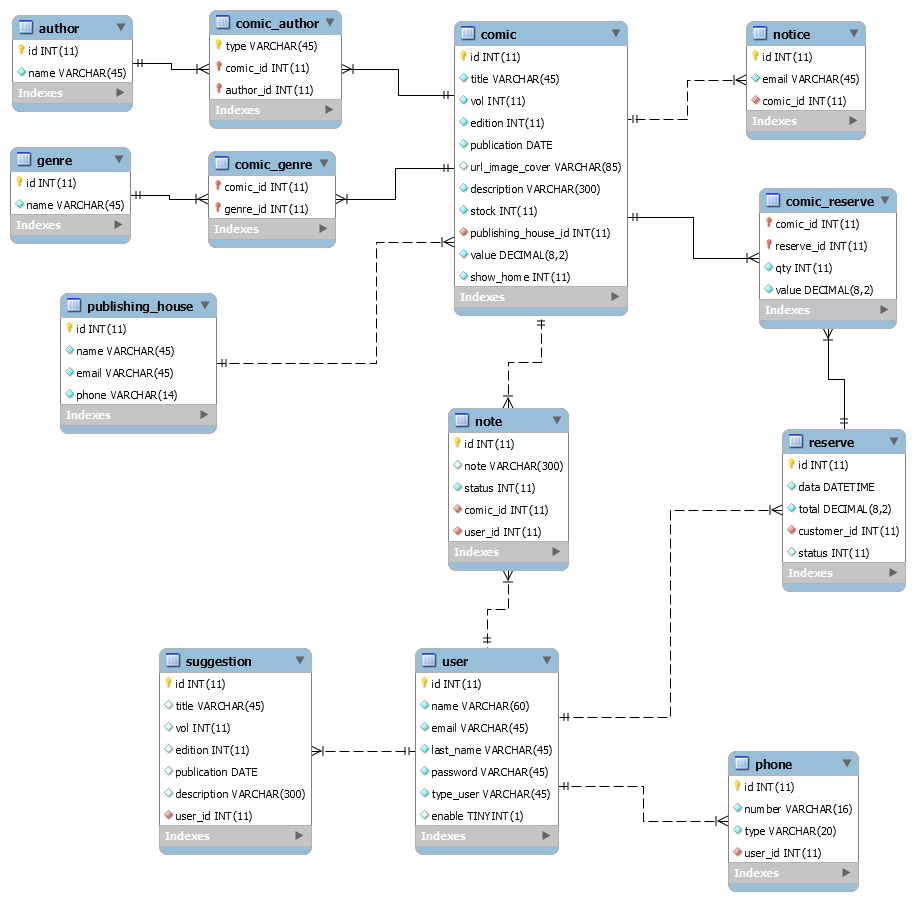


Figura 5 Diagramma entità relazione

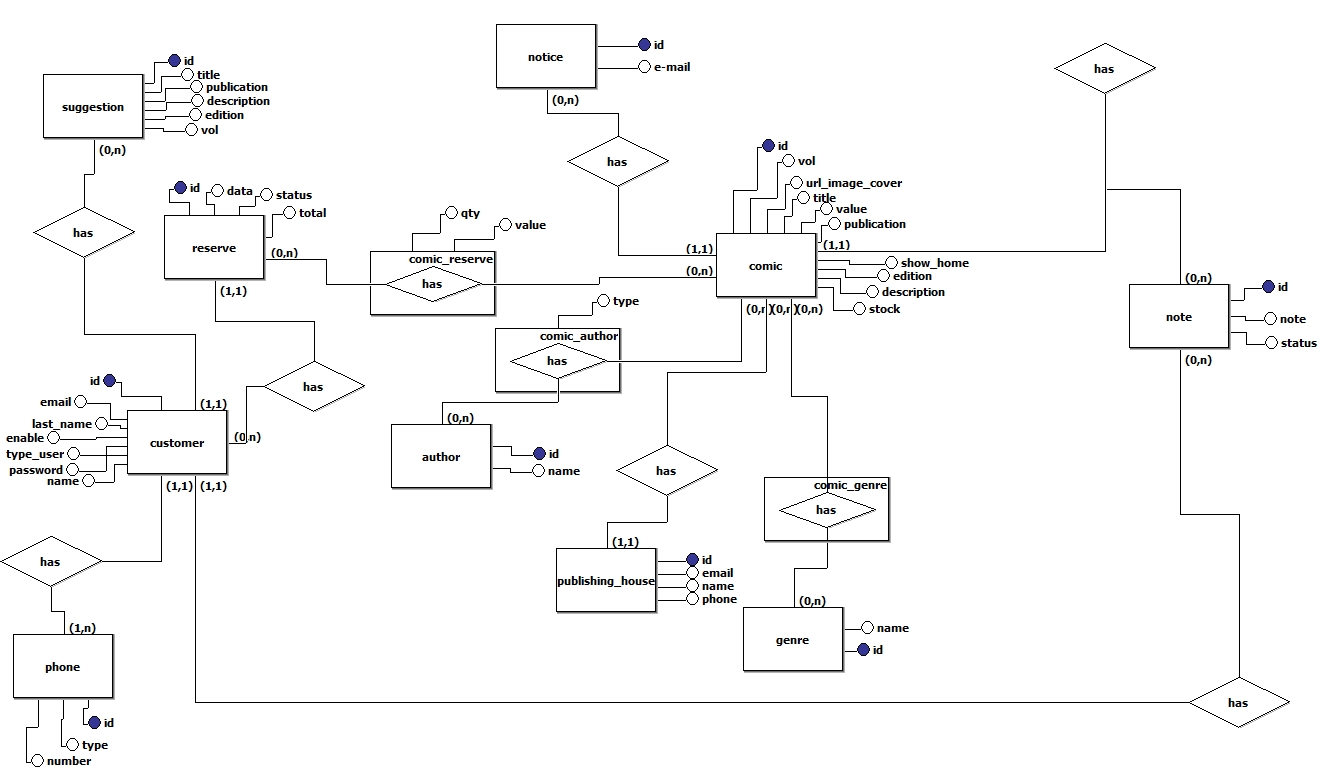


Figura 6 Modello entità relazione

1. Dizionario dei dati
   1. **Tabella author**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attributo** | **Tipo** | **Descrizione** | **Chiave Primaria** | **Chiave Esterna** | ***Not Null*** | ***Unique*** |
| Id | INT(11) | Numero identificatore | Si |  | Si | Si |
| Name | VARCHAR(45) | Nome dell’autore | No |  | Si | No |

* 1. **Tabella comic\_author**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attributo** | **Tipo** | **Descrizione** | **Chiave Primaria** | **Chiave Esterna** | ***Not Null*** | ***Unique*** |
| Type | VARCHAR(45) | Tipo dell’authore | Si |  | Si | No |
| Author\_id | INT(11) | Identificatore dell’autore | Si | Si | Si | No |
| Comic\_id | INT(11) | Identificatore del fumetto | Si | Si | Si | No |

* 1. **Tabella comic**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Attributo** | **Tipo** | **Descrizione** | **Chiave Primaria** | **Chiave Esterna** | ***Not Null*** | ***Unique*** |
| Id | INT(11) | Identificatore dell’autore | Si |  | Si | Si |

* 1. **Tabella genre**
  2. **Tabella comic\_genre**
  3. **Tabella notice**
  4. **Tabella reserve**
  5. **Tabella comic\_reserve**
  6. **Tabella publishing\_house**
  7. **Tabella user**
  8. **Tabella phone**
  9. **Tabella note**
  10. **Tabella suggestion**

1. prototipi