Università degli Studi di Salerno
Corso di Ingegneria del Software

BirdComics
System Design Document
Versione 3.0

BirdComics

Data: 23/03/2025

Coordinatore del progetto:

Nome	Matricola

Partecipanti:

Nome	Matricola
De Rosa Giuseppe	0512110683
Buccino Adriano	0512120484

Scritto da:	

Revision History

Data	Versione	Descrizione	Autore
23/03/2025	3.0	Modifica sottosistemi e matrice controllo degli accessi	De Rosa Giuseppe Buccino Adriano
07/01/2024	2.0	Riorganizzazione del documento con ridefinizione dei sottosistemi e del relativo hw/sw mapping	De Rosa Giuseppe Buccino Adriano
24/11/2024	1.0	Prima Versione del System Design Document	De Rosa Giuseppe Buccino Adriano

1. Introduzione

1.1. Scopo del sistema

Il sistema ha lo scopo di sviluppare e gestire una piattaforma di e-commerce dedicata alla vendita di fumetti, denominata *BirdComics*. La creazione di un e-commerce focalizzato sulla vendita di fumetti, si rivela una risposta appropriata alle necessità di un mercato in continua crescita. Differenziandosi nettamente da un negozio fisico. Un aspetto fondamentale di questa iniziativa è l'accessibilità: un e-commerce è aperto 24 ore su 24, 7 giorni su 7, permettendo ai consumatori di effettuare acquisti in qualsiasi momento, senza le restrizioni di orario imposte dai negozi tradizionali. Questa piattaforma permette agli utenti di acquistare fumetti in formato cartaceo, fornendo maggiore flessibilità e comodità esplorando una vasta selezione di titoli (dai grandi classici ai fumetti in edizione limitata o rari da trovare) che un negozio online può offrire rispetto a uno spazio fisico, il quale è limitato dalla capacità espositiva. L'obiettivo principale è offrire un'esperienza di acquisto comoda, accessibile e personalizzata, raggiungendo un pubblico globale e soddisfacendo le crescenti esigenze dei lettori di fumetti. Inoltre, il sistema dovrà essere altamente scalabile, permettendo di gestire un ampio catalogo di prodotti, utenti e transazioni senza compromettere le performance.

1.2. Obiettivi di progettazione

1.2.1. Prestazioni

Tempo di risposta	Il sistema deve rispondere a ogni richiesta dell'utente (come la navigazione o l'acquisto) entro 2 secondi (**RNF7**).
Throughput	Il sistema deve supportare almeno 1000 utenti simultanei senza degradare le prestazioni (**RNF8**).
Memoria	L'uso della memoria deve essere ottimizzato per mantenere prestazioni

	elevate garantendo il tempo di risposta ed il throughput sopra citati
Velocità ricerca	I risultati della ricerca nel catalogo prodotti devono essere aggiornati in tempo reale senza ricaricare la pagina (**RNF9**).

1.2.2. Garanzia di funzionamento

Robustezza	Il sistema deve resistere a input non validi e a comportamenti imprevisti degli utenti senza andare in crash.
Affidabilità	Il sistema deve garantire un funzionamento stabile, con una disponibilità del 99.9% (**RFN5**).
Disponibilità	Il sito deve essere operativo quasi costantemente, con downtime ridotto al minimo per manutenzione
Tolleranza ai guasti	Deve continuare a funzionare anche in caso di errori, ad esempio grazie a server di backup
Sicurezza	Protezione contro accessi non autorizzati (**RNF1**), password crittografate (**RNF2**), prevenzione di attacchi SQL Injection (**RNF3**).
Sicurezza utenti	Protezione dei dati personali degli utenti e delle transazioni di pagamento.
Sicurezza catalogo	Gli aggiornamenti al catalogo e alle scorte devono essere immediati (**RFN6**)

1.2.3. Manutenzione

Estendibilità	Facilità nell'aggiungere nuove funzionalità (come l'integrazione con nuovi gateway di pagamento).
Modificabilità	Possibilità di modificare funzionalità

	esistenti senza influire negativamente sul sistema.
Adattabilità	Capacità di adattarsi a diversi contesti o domini (ad es. un altro paese con diverse normative).
Leggibilità	Codice ben documentato e strutturato per facilitare la comprensione e il debug (**RNF10**).
Tracciabilità dei requisiti	Possibilità di mappare facilmente il codice alle specifiche originali del progetto.

1.2.4. Utente finale

Utilità	Il sistema deve supportare gli utenti nel completare i loro acquisti, gestire i carrelli, e seguire i processi di pagamento in modo semplice.
Usabilità	Il sistema deve essere intuitivo, con interfaccia user-friendly
Feedback	Il sistema deve fornire messaggi di conferma e notifiche degli errori in tempo reale (**RNF4**).
Accessibilità	Deve essere accessibile su vari dispositivi, come desktop, tablet e smartphone.

1.2.5. Compromessi

Spazio vs. Velocità	Potrebbe essere necessario utilizzare più memoria per ottenere tempi di risposta più rapidi
Tempo di consegna vs. Funzionalità	Ridurre le funzionalità per rispettare le scadenze di lancio del sito
Sicurezza vs. Usabilità	Aumentare la sicurezza (ad esempio, richiedendo password complesse e autenticazione a due fattori) può rendere il sistema meno intuitivo e creare

	frustrazione per gli utenti che cercano un'esperienza rapida e fluida.

1.3. Definizioni, acronimi e abbreviazioni

RNF: Requisito Non Funzionale
RAD: Requirements Analysis Document

1.4. Riferimenti

Nel corso del documento facciamo riferimento ai Requisiti Non Funzionali, agli Use Cases e al Class Diagram presenti nel documento RAD.

1.5. Panoramica

Nei paragrafi successivi presentiamo l'analisi architetturale della piattaforma BirdComics. La sua decomposizione in sottosistemi, strategie di deployment su hardware, strategie per la gestione dei dati persistenti ed eventuali Boundary Condition del sistema.

2. Architettura software proposta

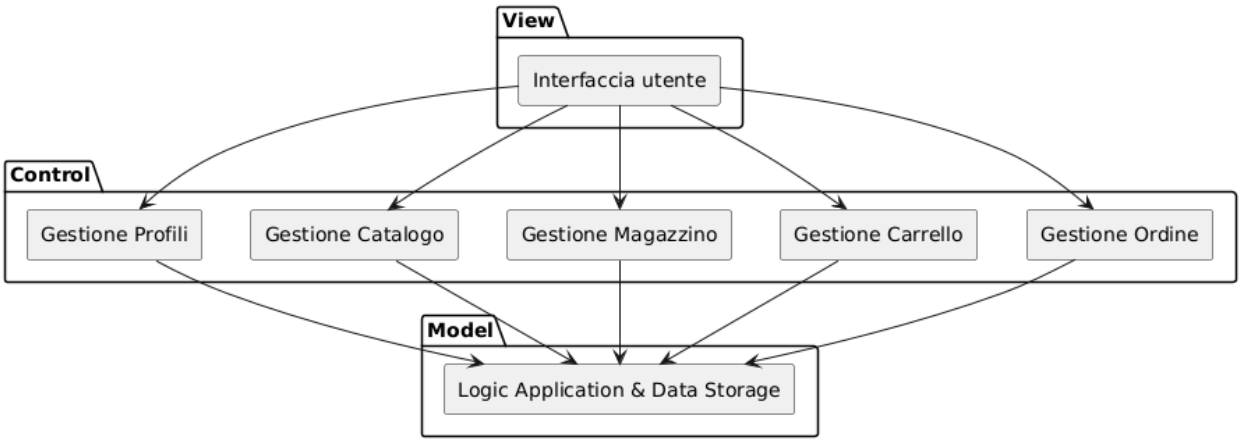
2.1. Decomposizione del sottoproblema

Il sistema BirdComics è sviluppato e diviso secondo questi modelli:

- **Server TomCat**: l'applicativo utilizzerà il modello MVC (Model View Control). Nella modifica delle informazioni, la view passa le informazioni al Control e quest'ultimo per salvare le informazioni di modifica, passa le informazioni al model in maniera diretta. Il model sarà diviso in sottosistemi, dove ogni sottosistema avrà un ruolo specifico con basso accoppiamento tra loro, inoltre saranno implementati tramite stile "facade".
- **Client**: Il client si interfaccerà unicamente con la View del sistema Tomcat, la comunicazione sarà di tipo client-server utilizzando il protocollo https
- **Database**: il sistema di Model interagisce con il database tramite il DAO Pattern (Data Access Object). Per farlo, nel Model verrà creata un'interfaccia DAO, che viene invocata da tutti i sottosistemi che necessitano di interagire con il database. Ogni tipo di oggetto avrà il suo DAO, che sarà implementato da un componente che

conosce il tipo specifico di database da utilizzare. L'oggetto DAO fornirà i metodi necessari per eseguire operazioni di persistenza come la creazione, la lettura, l'aggiornamento e la cancellazione (CRUD) dei dati. L'implementazione concreta del DAO avrà la logica di connessione e interazione con il database specifico. Utilizzando il DAO Pattern, possiamo garantire un basso accoppiamento tra il modello e il sistema di persistenza. In altre parole, se vogliamo cambiare il database sottostante o la modalità di accesso ai dati, dobbiamo solo creare una nuova implementazione del DAO, senza necessitare di modificare i sottosistemi esistenti che interagiscono con il modello.

Component Diagram



Sottosistema - Gestione Profilo

Il sottosistema si occupa di tutte le operazioni relative alla creazione, modifica, cancellazione e gestione dei profili degli utenti all'interno del sistema. Gli utenti possono essere di diverso tipo: Ospiti, Clienti, Dipendenti, Risorse Umane, Gestori di Magazzino e Direttori Generali. Ogni tipo di utente dispone di specifici permessi e può eseguire determinate operazioni consentite.

Servizi e Operazioni

Servizio: Registrazione Utenti

Descrizione: Questo servizio permette la registrazione di nuovi account per diversi ruoli, garantendo l'accesso al sistema in base al tipo di utente.
Operazioni:

- registraAccount()
    - Registra un nuovo account nel sistema. Il tipo di account è determinato dai parametri passati.
    - Ruoli che possono chiamare questo metodo:
        - Ospite: Può registrare un account Cliente.
        - Risorse Umane: Può registrare account per i seguenti ruoli:
            - Finanza
            - Magazziniere
            - Operatore Logistico
            - Gestore Catalogo
            - Assistenza
        - Direttore Magazzino: Può registrare un account Risorse Umane.
        - Direttore Generale: Può registrare un account Gestore Magazzino.

Servizio: Modifica Profilo Utente
Descrizione: Consente la modifica delle informazioni personali e professionali degli utenti, con limiti definiti per ciascun ruolo.
Operazioni:

- modificaPassword()
    - Modifica la password dell'utente.
    - Ruoli che possono chiamare questo metodo: Tutti i ruoli.
- modificaInfo()
    - Modifica le informazioni del profilo utente. Le informazioni modificabili dipendono dal ruolo dell'utente che effettua la modifica.
    - Ruoli che possono chiamare questo metodo:
        - Cliente: Può modificare il proprio profilo.
        - Risorse Umane: Può modificare i profili dei seguenti ruoli:
            - Finanza
            - Magazziniere
            - Operatore Logistico
            - Gestore Catalogo
            - Assistenza
        - Direttore Magazzino: Può modificare i profili delle Risorse Umane.
        - Direttore Generale: Può modificare i profili dei Gestori di Magazzino e il proprio profilo.

Servizio: Cancellazione Utenti
Descrizione: Gestisce la rimozione di account dal sistema, con permessi specifici per i diversi ruoli.

Operazioni:

- rimuoviAccount()
    - Rimuove un account dal sistema. Il tipo di account che può essere rimosso dipende dal ruolo dell'utente che effettua la cancellazione.
    - Ruoli che possono chiamare questo metodo:
        - Cliente: Può cancellare il proprio account.
        - Risorse Umane: Può cancellare account per i seguenti ruoli:
            - Finanza
            - Magazziniere
            - Operatore Logistico
            - Gestore Catalogo
            - Assistenza
        - Direttore Magazzino: Può cancellare account Risorse Umane.
        - Direttore Generale: Può cancellare account Gestori di Magazzino.

Servizio: Autenticazione e Disconnessione
Descrizione: Gestisce l'accesso e la chiusura delle sessioni degli utenti.
Operazioni:

- login()
    - Verifica le credenziali dell'utente e permette l'accesso al sistema (Tutti i ruoli)
- logout()
    - Chiude la sessione dell'utente (Tutti i ruoli)

Servizio: Consultazione Profili
Descrizione: Permette di visualizzare i dettagli dei profili utente.
Operazioni:

- getUserDetails()
    - Restituisce i dettagli di un utente specifico.
    - Ruoli che possono chiamare questo metodo:
        - Risorse Umane: Può consultare i profili dei dipendenti.
        - Direttore Magazzino: Può consultare i profili delle Risorse Umane.
        - Direttore Generale: Può consultare tutti i profili.
- getUsersByRole()
    - Restituisce una lista di utenti filtrata per ruolo.
    - Ruoli che possono chiamare questo metodo:
        - Risorse Umane: Può consultare i profili dei dipendenti.
        - Direttore Magazzino: Può consultare i profili delle Risorse Umane.
        - Direttore Generale: Può consultare tutti i profili.

Sottosistema - Gestione del Catalogo

Il sottosistema si occupa di tutte le operazioni relative alla gestione e consultazione dei fumetti presenti nel sistema. Ospiti e Clienti possono consultare e ricercare il catalogo, mentre il Gestore Catalogo può amministrare il catalogo, gestendo i fumetti e accedendo ai dettagli per verificarne la correttezza.

Servizi e Operazioni

Servizio: Gestione dei Fumetti
Descrizione: Consente di aggiungere, modificare o rimuovere i fumetti presenti nel catalogo.
Operazioni:

- addFumetto()
    - Aggiunge un nuovo fumetto al catalogo (Gestore Catalogo)
- rmFumetto()
    - Rimuove un fumetto dal catalogo (Gestore Catalogo)

Servizio: Ricerca nel Catalogo
Descrizione: Consente a tutti gli utenti non amministrativi di esplorare il catalogo, trovando i fumetti desiderati tramite ricerca.
Operazioni:

- ricercaTitolo()
    - Ricerca un fumetto solo per titolo (Cliente, Ospite)
- ricercaGenere()
    - Ricerca un fumetto solo per genere (Cliente, Ospite)
- ricercaID()
    - Ricerca un fumetto per titolo, ID o genere (Gestore Catalogo)
- visualizzaCatalogo()
    - Mostra l'intero catalogo di fumetti (Cliente, Ospite)

Servizio: Visualizzazione Dettagli
Descrizione: Fornisce i dettagli di un fumetto selezionato. Questo servizio è utile sia per i clienti e gli ospiti, sia per il Gestore Catalogo che ne verifica le informazioni.
Operazioni:

- visualizzaDettagli()

    - Mostra le informazioni complete di un fumetto, inclusi titolo, genere, prezzo, descrizione(Gestore Catalogo, Cliente, Ospite)

Sottosistema - Gestione Magazzino

Il sottosistema si occupa della gestione del magazzino fisico, inclusi scaffali, posizionamento dei fumetti e risorse disponibili. Permette ai Magazzinieri di gestire i fumetti all'interno degli scaffali a loro assegnati, mentre i Direttori di Magazzino gestiscono le operazioni relative agli scaffali e agli spazi del magazzino a loro associato.

Servizio: Gestione Scaffali
Descrizione: Gestisce l'organizzazione dei fumetti sugli scaffali nel magazzino.
Operazioni:

- addFumettoScaffale()
    - Aggiunge un fumetto a uno scaffale (Magazziniere)
- modifyQuantitaFumetto()
    - Modifica la quantità di un fumetto su uno scaffale (Magazziniere)
- rmFumettoScaffale()
    - Rimuove un fumetto da uno scaffale (Magazziniere)
- getScaffaleMagazzino()
    - Restituisce la lista degli scaffali di un magazzino a cui è assegnato l'utente (Magazziniere)

Servizio: Organizzazione Magazzino
Descrizione: Permette ai Direttori di gestire le risorse fisiche e lo staff dei magazzini.
Operazioni:

- addMagazzino()
    - Aggiunge un nuovo magazzino (Direttore Generale)
- rmMagazzino()
    - Rimuove un magazzino esistente (Direttore Generale)
- addScaffale()
    - Aggiunge scaffali a un magazzino (Direttore Magazzino)
- rmScaffale()
    - Rimuove scaffali da un magazzino (Direttore Magazzino)
- getAllMagazzini()
    - Restituisce la lista di tutti i magazzini presenti nel sistema (Direttore Magazzino)

Sottosistema - Gestione Carrello

Il sottosistema gestisce tutte le operazioni relative alla gestione dei fumetti che un utente desidera acquistare. Consente di aggiungere, rimuovere, modificare la quantità dei fumetti nel carrello, calcolare il totale dell'ordine e caricare il carrello dal database.

Servizio: Gestione Carrello

Descrizione: Questo servizio gestisce le operazioni all'interno del carrello, come l'aggiunta, la rimozione, la modifica delle quantità dei fumetti, il calcolo del totale e il caricamento del carrello dal database.

Operazioni:

1. aggiungiFumetto()
    - Aggiunge un fumetto al carrello con una quantità specificata e verifica la disponibilità del prodotto prima di aggiungerlo. (Cliente)
2. rimuoviFumetto()
    - Rimuove un fumetto dal carrello (Cliente)
3. modificaQuantita()
    - Modifica la quantità di un fumetto presente nel carrello (Cliente)
4. svuotaCarrello()
    - Svuota completamente il carrello (Cliente)
5. visualizzaCarrello()
    - Restituisce la lista degli elementi presenti nel carrello (Cliente)
6. visualizzaProdottiCarrello()
    - Restituisce i dettagli dei prodotti presenti nel carrello (Cliente)
7. calculateTotalAmount()
    - Calcola il totale dell'ordine in base ai prodotti presenti nel carrello (Cliente)
8. loadCart()
    - Carica il carrello di un utente (Cliente)

Sottosistema - Gestione Ordine

Il sottosistema gestisce tutte le operazioni legate agli acquisti, inclusa la creazione, il monitoraggio dello stato e la gestione del tracking della spedizione.

Servizio: Gestione Ordini

Descrizione:

Questo servizio gestisce tutte le operazioni relative agli ordini, tra cui la creazione, la visualizzazione e la gestione dello stato e del tracking.

Operazioni:

- creaOrdine()
    - Descrizione: Crea un nuovo ordine a partire dagli articoli nel carrello, avviando il processo di acquisto. (Cliente)
- visualizzaStatoOrdine()
    - Descrizione: Visualizza lo stato attuale dell'ordine, che può essere "spedito" o "non spedito", e mostra il tracking, se disponibile. (Cliente)
- getOrdiniNonSpediti()
    - Descrizione: Recupera tutti gli ordini che non sono ancora stati spediti. Utile per la gestione interna e per il monitoraggio degli ordini in attesa di spedizione. (Operatore Logistico)
- getOrdiniPerUtente()
    - Descrizione: Recupera tutti gli ordini effettuati da un utente specifico, identificato tramite la sua email. (Cliente)

Servizio: Gestione Tracking

Descrizione:

Permette di aggiungere e modificare le informazioni relative al tracking della spedizione dell'ordine.

Operazioni:

1. aggiungiTracking()
    - Descrizione: Aggiunge il numero di tracking a un ordine quando è spedito, consentendo il monitoraggio della spedizione. (Operatore Logistico)
2. modificaTracking()
    - Descrizione: Modifica il numero di tracking esistente di un ordine, se necessario. (Operatore Logistico)

Servizio: Fatturazione e Pagamento

Descrizione:

Gestisce le operazioni relative alla fatturazione e al pagamento degli ordini, utilizzando PayPal come metodo di pagamento.

Operazioni:

1. generareFattura()
    - Descrizione: Genera una fattura per l'ordine, con dettagli sui fumetti acquistati, quantità, prezzi e totale. (Cliente)
2. processaPagamento()

    - Descrizione: Gestisce l'elaborazione del pagamento tramite PayPal, eseguendo il pagamento autorizzato e confermando l'avvenuta transazione. (Cliente)
3. authorizePayment()
    - Descrizione: Autorizza il pagamento tramite PayPal, creando una richiesta di pagamento e restituendo un link di approvazione per il cliente. (Cliente)
4. getPaymentDetails()
    - Descrizione: Recupera i dettagli di un pagamento specifico, inclusi lo stato, le transazioni e le informazioni sul pagante. Questo metodo è utilizzato per verificare lo stato di un pagamento dopo l'approvazione del cliente. (Cliente)

Sottosistema Assistenza

Descrizione:
Gestisce le richieste di assistenza clienti, permettendo di cercare ordini e visualizzare fatture.

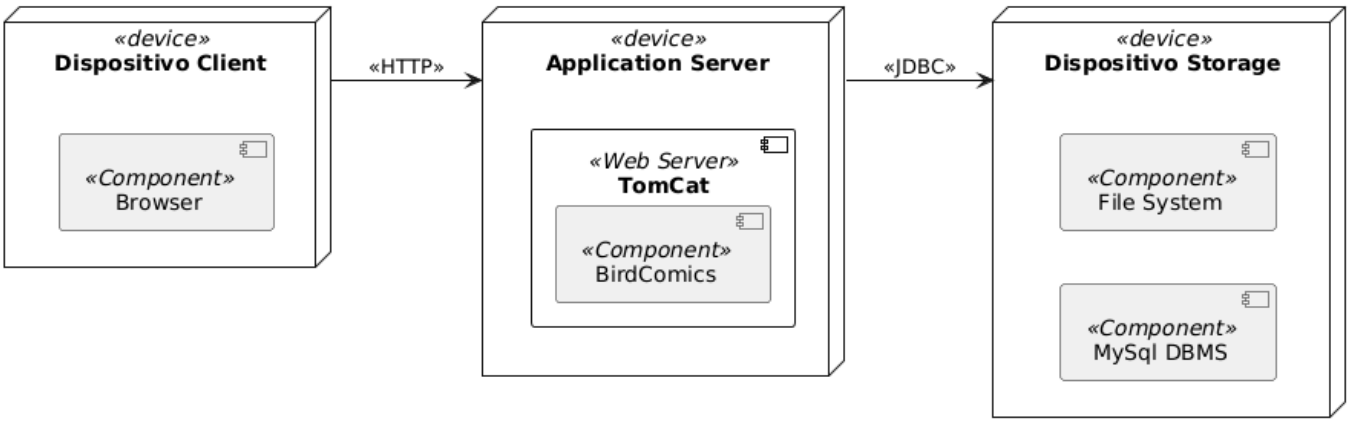
Sottosistema Finanza

Descrizione:
Gestisce le operazioni finanziarie, come la visualizzazione delle fatture in un intervallo di tempo.

2.2. Mappatura Hardware e Software

Il sistema distribuito di BirdComics non è stato progettato per girare su una configurazione hardware specifica, ma bensì su una configurazione software specifica. Gli host possono essere costruiti con un ampio parco hardware, purché sia architettura x86 a 64 bit con la possibilità di differenziare l’hardware di ogni singolo host. La configurazione software di ogni host è un sistema operativo che permette di distribuire la potenza di calcolo, proxmox, con installato al suo interno una macchina virtuale linux contenente tomcat versione 9.

Deployment Diagram



2.3. Gestione dati persistenti

La gestione dei dati persistenti è fondamentale in un sistema distribuito. Rendere i dati persistenti, ovvero archiviati su memoria di massa non volatile, permette ai sottosistemi in esecuzione su VM o su Host fisici sparsi per la rete, di accedere ad un archivio, che sia esso centrale o distribuito, garantisce che le informazioni siano conservate e recuperabili in qualsiasi momento.
L'archivio che andremo a realizzare per BirdComics sarà separato dal sistema distribuito TomCat e gestito da nodi specializzati alla sola archiviazione dei dati. L'archiviazione è resa possibile tramite i servizi di database e Network Attached Storage installati su diversi nodi fisici, distribuiti geograficamente.
Per la gestione dei database utilizziamo il server MySql, che tratterà i dati di tipo carattere, stringhe, interi, ecc… mentre per l'archiviazione dei file multimediali, come immagini, pdf, ecc… si utilizzerà un sistema di Network Attached Storage.
Implementando queste tecnologie esterne al sistema TomCat, garantiamo come caratteristiche minori, che al riavvio del sistema, i dati non debbano essere ricaricati dall'attore in quanto "sopravvissuti" al riavvio.
Le caratteristiche più importanti includono la distribuzione geografica dei dati, che consente ai clienti di accedervi più rapidamente grazie alla riduzione della distanza fisica (numero di hop), la gestione dell'accesso concorrente tramite i database, la duplicazione dei dati su più hop, la scalabilità attraverso l'implementazione di ulteriori host e la sicurezza.

Il sistema sarà implementato in questo modo, nelle prime fasi iniziali, per ridurre il costo di realizzazione iniziale e dato che i clienti non saranno molti da far rallentare il sistema, ed i fumetti non saranno molti si è deciso di integrare le immagini all'interno del server TomCat. Una volta che il sito amplierà il catalogo ed i clienti aumenteranno, si passerà ad un sistema di gestione immagini esterno NAS.

2.4. Controllo degli Accessi e Sicurezza

Attore / Oggetti	Gestione Profilo	Gestione Catalogo	Gestione Magazzino	Gestione Carrello	Gestione Ordine
Ospite	registraAccount()	ricercaGenere(), ricercaTitolo(), verificaQty(), getFumetti(), visualizzaCatalogo(), visualizzaDettagli()	-	-	-
Cliente	getIndirizzo(), setIndirizzo(), getInfo(), login(), logout(), setInfo()	ricercaGenere(), ricercaTitolo(), verificaQty(), getFumetti(), visualizzaCatalogo(), visualizzaDettagli()	-	aggiungiAlCarrello(), getFumettiCarrello(), modificaQtyCarrello(), delFumettoCarrello(), svuotaCarrello(), visualizzaCarrello(), calculateTotalAmount(), loadCart()	getFattura(), effettuaAcquisto(), creaOrdine(), visualizzaStatoOrdine(), getOrdiniPerUtente(), generareFattura(), processaPagamento(), authorizePayment(), getPaymentDetails()

Gestore Generale	getIndirizzo(), setInfoGestoreMagazzino(), getInfo(), login(), logout(), insertGestoreMagazzino(), delGestoreMagazzino(), getGestoriMagazzino()	-	insertSede(), getSedi(), deleteSede()	-	-
Gestore Magazzino	getIndirizzo(), setInfoRisorseUmane(), getInfo(), login(), logout(), insertRisorseUmane(), delRisorseUmane(), getRisorseUmane()	-	insertScaffale(), deleteScaffale(), getScaffali()	-	-
Risorse Umane	getIndirizzo(), getInfo(), login(), logout(), setInfoDipendenti(), insertDipendente(), delDipendente(), getDipendenti()	-	-	-	-
Magazziniere	getIndirizzo(), getInfo(), login(), logout()	-	getLavori(), updateLavoro(), getFumetti(), getScaffaliLiberi(), addFumettoAScaffale(), delFumettoAScaffale()	-	-

Gestore Catalogo	getIndirizzo(), getInfo(), login(), logout()	ricercaGenere(), ricercaTItolo(), ricercaID(), insertFumetto(), deleteFumetto(), updateFumetto(), getFumetti()	addMansione Magazzinieri()	-	-
Operatore Logistico	getIndirizzo(), getInfo(), login(), logout()	getFumetti()	-	-	ordiniNoTracking(), insertTracciamento()

2.5. Controllo Software Globale

Il sistema BirdComics adotta un'architettura di tipo event-driven, in cui le richieste degli utenti provenienti dal client browser, vengono inizializzate tramite interazioni con l'interfaccia grafica. Quando un utente effettua un'azione, come cliccare su un fumetto per acquistarlo o aggiornare il proprio profilo, viene generato un evento che interagisce con il layer VIEW presente sui server BirdComics. Per ogni evento generato dall'utente, esiste un rispettivo listener sul sistema che innesca un processo di elaborazione. L'elaborazione delle richieste da parte della View si traduce, in un cambio di pagina o in un cambio di stato dei dati. Nel caso di cambio pagina, la view interagisce con il layer control il quale provvede a reindirizzare l'attore sulla pagina richiesta. Nel caso di cambio consistente dei dati (aggiungi al carrello, modifica profilo, effettua acquisto, etc…) la View riceve i dati inseriti da parte dell'attore, li inoltra al Control e poi questi dati vengono inoltrati al Model il quale esegue la logica applicativa e aggiorna i dati persistenti. Come è possibile intuire sia il Control che il Model ricevono eventi dagli strati a loro collegati. Per il caricamento e la visualizzazione delle informazioni persistenti, Il Model carica i dati e li inoltra alla View

2.6. Boundary Condition

Le condizioni boundary identificate per il nostro e-commerce includono situazioni legate all'avvio del sistema, il suo spegnimento, il riavvio in seguito a un fallimento.

2.6.1. Primo avvio

Prima del primo avvio, lo sviluppatore che sta configurando il sistema dovrà inserire manualmente almeno un profilo Direttore Generale all'interno del database MySQL, dopo aver avviato il relativo servizio. Questo è necessario perché, nella nostra piattaforma, solo un amministratore può registrare nuovi amministratori o gestori.
Completata questa operazione, si potrà avviare il web server che ospita l'e-commerce.

Per gli avvii successivi, sarà sufficiente avviare il DBMS e il web server.

2.6.2. Avvio in caso di fallimento

In caso di un'interruzione improvvisa del sistema, il database utilizza transazioni per garantire la consistenza delle informazioni. Eventuali modifiche parziali in corso non verranno salvate, evitando inconsistenze nei dati.
Non sono previste funzionalità per il ripristino dello stato precedente all'arresto improvviso.

2.6.3. Spegnimento

Per spegnere correttamente il sistema, sarà sufficiente terminare l'esecuzione del web server. Poiché il web server gestisce anche la connessione al DBMS, non sarà necessario spegnere manualmente il database.
Eventuali transazioni non completate verranno annullate, garantendo la consistenza dei dati persistenti.

2.6.4. Errato accesso ai dati persistenti

Se si verificano errori nella configurazione del database o nell'utilizzo del driver per connettersi ad esso, il sistema mostrerà una pagina di errore personalizzata. Questa pagina fornirà dettagli sulla causa e sul punto preciso del codice sorgente in cui si è verificato il problema.
Lo sviluppatore dovrà correggere il problema prima di riavviare il sistema. Una volta risolto, il sistema tornerà operativo come di norma.