**Università degli Studi di Salerno**

**Corso di Ingegneria del Software**

**EdenJewelry SDD\_EdenJewelry**

**Versione 2.4**

Data: 02/02/2025

|  |  |
| --- | --- |
| Progetto: EdenJewelry | Versione: 2.4 |
| Documento: SDD\_EdenJewelry | Data: 02/02/2025 |

## Coordinatore del progetto:

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
|  |  |
|  |  |

**Partecipanti:**

|  |  |
| --- | --- |
| **Nome** | **Matricola** |
| Alessandro Di Palma | 0512115939 |
| Gaetano D’Alessio | 0512110836 |
| Luigi Montuori | 0512117799 |
| Miriam Eva De Santis | 0512117121 |

Luigi Montuori

**Scritto da:**

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Versione** | **Descrizione** | **Autore** |
| 17/11/2024 | 1.0 | Prima stesura del file di System Design | Luigi Montuori |
| 18/11/2024 | 1.1 | Continuo criteri di affidabilità | Luigi Montuori |
| 20/11/2024 | 1.2 | Scritti capitoli 1 e 2 | Gaetano D’Alessio |
| 21/11/2024 | 1.3 | Scritti 3.4, 3.5 e 3.6 | Luigi Montuori |
| 21/11/2024 | 1.4 | Fine stesura 3.1 | Gaetano D’Alessio |
| 21/11/2024 | 1.5 | Aggiunta Deployment Diagram | Luigi Montuori |
| 21/11/2024 | 1.6 | Mapping hardware/software | Miriam Eva De Santis |
| 23/11/2024 | 1.7 | Lavori ultimati sul file | Gaetano D’Alessio |
| 23/11/2024 | 1.8 | Correzione dell’indice | Alessandro Di Palma |
| 21/12/2024 | 1.9 | Rimozione Boundary Condition | Luigi Montuori |
| 22/12/2024 | 2.0 | Modifiche decomposizione in sottosistemi | Luigi Montuori |
| 03/01/2025 | 2.1 | Aggiunta servizi offerti | Luigi Montuori |
| 03/01/2025 | 2.2 | Correzione par. 3.3 e 3.4 | Miriam Eva De Santis |
| 03/01/2025 | 2.3 | Ultime modifiche | Gaetano D’Alessio |

|  |  |  |
| --- | --- | --- |
|  | Ingegneria del Software | Pagina 2 di 14 |

|  |  |
| --- | --- |
| Progetto: EdenJewelry | Versione: 2.4 |
| Documento: SDD\_EdenJewelry | Data: 02/02/2025 |

|  |  |  |  |
| --- | --- | --- | --- |
| 02/02/2025 | 2.4 | Ulteriori correzioni | Gaetano D’Alessio |

|  |  |  |
| --- | --- | --- |
|  | Ingegneria del Software | Pagina 3 di 14 |

**Indice**

1. INTRODUZIONE 5
   1. Scopo del sistema 5
   2. Obiettivi di sistema 5

|  |  |  |
| --- | --- | --- |
| 1.2.1. | Criteri di prestazione | 5 |
| 1.2.2. | Criteri di affidabilità | 5 |
| 1.2.3 | Criteri di costo | 6 |
| 1.2.4 | Criteri di mantenimento | 6 |
| 1.2.5 | Criteri di End User | 6 |
| 1.2.6 | Trade-off tra obiettivi | 6 |

1. Architettura software simili 6
2. [Architettura del sistema proposto 7](#_TOC_250009)
   1. [Decomposizione in sottosistemi 7](#_TOC_250008)
   2. [Mapping Hardware/Software 8](#_TOC_250007)
   3. [Gestione della persistenza 9](#_TOC_250006)
   4. [Controllo degli accessi 11](#_TOC_250005)
   5. [Controllo globale del Software 11](#_TOC_250004)
3. [Subsystem services 11](#_TOC_250003)
   1. [Presentation Layer 11](#_TOC_250002)
   2. [Application Layer 12](#_TOC_250001)
   3. [Data management layer 13](#_TOC_250000)
4. **INTRODUZIONE**

# Scopo del sistema

L’e-shop *EdenJewelry* si propone di offrire una soluzione che venga incontro alle esigenze dei clienti e dei venditori.

I primi cercano un catalogo ampio da consultare e vogliono la comodità degli shop online, per comprare i loro gioielli preferiti. I secondi, necessitano di una piattaforma per vendere i propri prodotti e interfacciarsi con i clienti.

I progettisti, quindi, si trovano a dover effettuare una scelta tra le duplici esigenze degli attori.

Alla fine, si è scelto di dare priorità alle richieste dei venditori. Lo scopo del sistema è allora quello di fornire un e-shop di gioielli con tutte le funzionalità che essi ritengono rilevanti.

# Obiettivi di progettazione

**Definizione delle priorità**

1. Priorità alta: funzionalità fondamentale, che necessita di essere implementata fin dalla prima release.
2. Priorità media: funzionalità meno rilevante, che può essere implementata nelle successive release.
3. Priorità bassa: funzionalità facoltativa, non necessita di essere implementata.

# Criteri di prestazione

## Memoria

Tutti i dati riguardo gli utenti, i prodotti e gli ordini devono essere inseriti e conservati in un database relazionale.

**Priorità**: Alta.

## Concorrenza

Il sistema deve gestire diversi accessi in contemporanea.

**Priorità**: Alta

## Tempo di risposta

I tempi di attesa devono essere ridotti il più possibile.

**Priorità**: Bassa.

## Throughput

Il throughput deve essere abbastanza alto per soddisfare un carico sempre crescente.

**Priorità**: Alta.

# Criteri di affidabilità

## Sicurezza

Deve garantire l’accesso alle varie funzionalità soltanto agli utenti autorizzati (le funzionalità del venditore non devono essere accessibili da utenti normali) **Priorità**: Alta.

## Attendibilità

Il sistema deve essere in grado di garantire l’attendibilità dei propri servizi. (Quando va a buon fine un acquisto, oltre a garantire la disponibilità del prodotto va

anche aggiornato il dato per gli altri utenti)

**Priorità**: Media.

# Criteri di costo

## Stock del catalogo

E da prevedere un costo (hours/men) per inserire i prodotti esistenti nel catalogo.

**Priorità**: Alta.

## Manutenzione

Ci saranno degli sviluppatori addetti alla manutenzione del sito.

**Priorità**: Bassa.

## Sviluppo

Il costo dello sviluppo, finita la fase di progettazione, sarà congruo alle funzionalità offerte.

**Priorità**: Alta.

# Criteri di mantenimento

## Portabilità

È possibile fare il deploy del sistema sviluppato su qualunque macchina provvista del container “Apache Tomcat”.

**Priorità**: Alta.

# Criteri di End User

## Responsività

Il sistema deve essere visualizzabile su dispositivi differenti.

**Priorità**: Media.

# 1.2.6 Trade-off tra obiettivi

|  |  |
| --- | --- |
| Funzionalità Vs Usabilità | Abbiamo preferito creare uno shop semplice da usare, piuttosto che concentrarci su  implementare molte funzionalità ridondanti. |
| Costo Vs Robustezza | Un giusto compromesso tra un sito robusto, ma che non richieda troppo tempo per essere implementato. |
| Sviluppo rapido Vs Funzionalità | Favorire e facilitare lo sviluppo, riducendolo  ad una serie di funzionalità essenziali. |

1. **Architetture software simili**

Nel nostro caso non abbiamo un’architettura software esistente sulla quale basarci. Abbiamo quindi deciso di confrontarci con gli e-shop di gioielli attualmente presenti sul mercato.

I siti che sono stati valutati sono:

1. **Pandora**, marchio leader nel campo della bigiotteria e gioielli, anche di lusso;
2. **Marlù**, competitor che è riuscito a ricavarsi la sua nicchia di utenti negli anni.

Entrambi suddividono i prodotti in categorie, hanno un’area utente dedicata e un carrello

consultabile prima del check-out. Inoltre, troviamo anche una wishlist dove è possibile salvare i gioielli desiderati, in attesa di sconti o solo come promemoria. L’area utente è accessibile tramite

e-mail e password, proteggendo dati sensibili dai malintenzionati.

Le tecnologie utilizzate sono eterogenee ma sicuramente a gestire la persistenza troviamo un database relazionale.

Queste sono tutte caratteristiche che consideriamo essenziali e abbiamo deciso di riportare nel nostro sistema.

# Architettura del sistema proposto

# Decomposizione in sottosistemi

La scelta per la decomposizione in sottosistemi è un’architettura client-server, stratificata attraverso tre layer.

I layer sono i seguenti:

* + - **Presentation layer**: contiene i componenti visibili all’utente e con cui egli interagisce;
    - **Application layer**: contiene la logica di business, regole di validazione e

gestione delle operazioni;

* + - **Data Management Layer**: contiene i componenti che interagiscono con il database e si occupano delle persistenze;

La scelta di questo pattern architetturale non è lasciata al caso. Non solo quest’ultimo è particolarmente adatto allo sviluppo di un sito di e-commerce, ma la decomposizione così effettuata tende a diminuire la complessità del sistema.

Quello che vogliamo ottenere è la massima coesione e il minimo accoppiamento.

La coesione misura la dipendenza tra le classi, mentre l’accoppiamento la dipendenza tra i sottosistemi.

Nello stile architetturale scelto la coesione è alta e i sottosistemi comunicano attraverso le loro interfacce.

La macchina virtuale è chiusa, cosiddetta opaca. Questo permette ai sottosistemi di

interagire tra di loro solo attraverso i servizi che espongono, senza lasciare che l’*upper* layer acceda al *bottom* layer.

Alla suddivisione strutturale in layer, ne segue una funzionale per package che rappresentano i sottosistemi scelti.

Conseguentemente, illustriamo i componenti appartenenti ai tre layers sopracitati.

## Presentation Layer:

* + AreaRiservata: si occupa di mostrare l’area dedicata alla registrazione e le aree riservate di utente e venditore;
  + Home: permette di visualizzare il catalogo e mostra i risultati della ricerca;
  + Acquisti: composto dalle pagine dinamiche relative alle fasi iniziali e finali di acquisto e scelta dei prodotti;

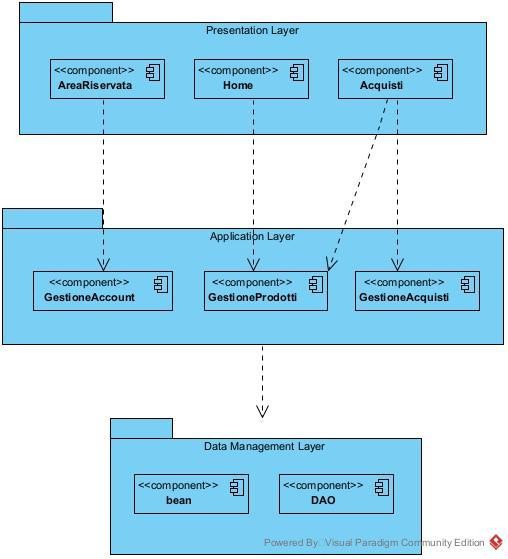
## Application Layer:

* + GestioneAccount: si occupa di gestire le funzionalità riguardanti gli utenti (autenticazione, registrazione);
  + GestioneProdotti: gestisce le funzionalità di aggiunta/rimozione prodotti e contiene la logica di business relativa alla ricerca dei prodotti;
  + GestioneAcquisti: gestisce la logica di business per salvare i prodotti nel carrello e nella wishlist (il carrello non essendo un dato persistente contiene anche operazioni di modifica dei dati), nonché l’operazione di acquisto.

## Data Management Layer:

* + bean: sono tutte le classi che rappresentano i dati persistenti a runtime,

praticamente una rappresentazione dei dati del database nell’ambiente di esecuzione;

* + DAO: tutte le classi che si occupano di aggiornare il database con i dati contenuti nei bean e recuperare informazioni dal database.

# Mapping Hardware/Software

Per gestire i vari aspetti del nostro shop (frontend, backend, persistenza), abbiamo deciso di affidarci ad una serie di tecnologie eterogenee.

Il sito è accessibile online tramite una connessione http al server, ossia la macchina fisica dove è stato installato e gira il sistema. Per poter fruire del sistema, il client interagisce con la parte

front-end tramite il browser. Per gestire la comunicazione tra client e server si utilizzano request/ response HTTP. Il deployment del sistema viene effettuato sul container Apache TomCat del host.

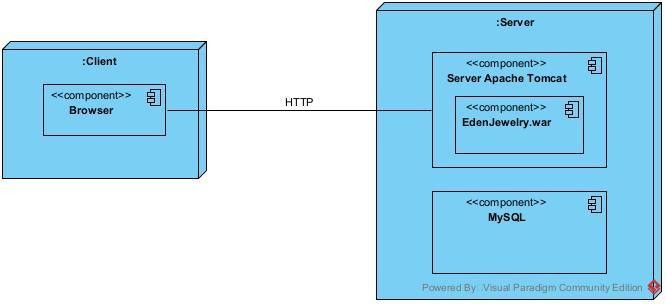
-Componenti hardware:

* + - Server (fisicamente una macchina x86):

-Componenti software:

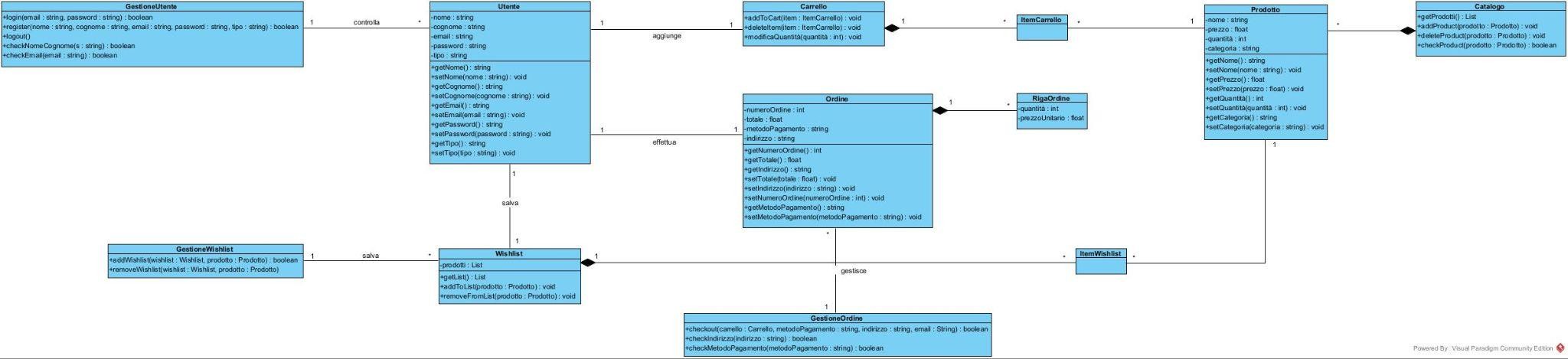
* + - Apache Tomcat: verrà utilizzato per la risoluzione del codice del backend e la gestione di request/response;
    - MySQL: per l'archiviazione di dati, principalmente persistenti (cioè necessari per un buon

funzionamento del sito). Il database viene ricavato dal modello EER;

* + - JavaScript: sfruttati per dare un contesto più interattivo e personalizzato all'utente specifico;
    - Jsp: permettono di aggiungere codice Java all'interno di file HTML per la generazione di contenuti dinamici;
    - Servlet: oggetto Java, residente lato server, che comunica con il client tramite request e response HTTP.
    - HTML: per lo "scheletro" del front end, che andremo poi a modificare con le JSP.
    - CSS: per personalizzare lo stile del sito e la parte front-end generale

# Gestione della persistenza

Come già detto poc’anzi, per la gestione della persistenza utilizzeremo un database relazionale. La scelta del DBMS è ricaduta su MySQL. Di seguito è presentata una raffinazione dell’object model, per meglio identificare le relazioni tra le classi/entità. Da notare l’aggiunta della signature dei metodi rispetto al class diagram presente nel RAD.



Da questa rappresentazione ricaviamo la tabella dei dati persistenti, con i relativi attributi.

## Utente:

* nome
* cognome
* email
* password
* tipo

## Prodotto:

* nome
* prezzo
* quantità
* categoria

## Ordine:

* numeroOrdine
* totale
* metodoPagamento
* indirizzo

## RigaOrdine:

* idRiga
* nomeProdotto
* numeroOrdine
* quantità
* prezzo

## Wishlist:

* idWishlist
* email

## ItemWishlist:

* idItem
* idWishlist
* nomeProdotto

Da questa tabella mancano ovviamente le classi di gestione, *Carrello* e *ItemCarrello*, perché non sono oggetti di cui abbiamo necessità di salvare la persistenza.

# Controllo degli accessi

|  |  |  |  |
| --- | --- | --- | --- |
|  | ATTORI | | |
| Utente | Utente non registrato | Venditore |
| GestioneAccount | Login() Logout() | Register() | Login() Logout() |
| GestioneProdotti | getProdotti () search() | getProdotti() search() | getProdotti() addProduct() deleteProduct() search() |
| GestioneAcquisti | addToCart() deleteItem() modificaQuantit a()  checkout() addWishlist() removeWishlist() |  | addToCart() deleteItem() modificaQuantit a()  checkout() addWishlist() removeWishlist() |

# Controllo globale del Software

Essendo un e-commerce, il controllo globale del software è centralizzato, per la precisione *event-driven*, il dispatcher si occuperà dello smistamento delle richieste http verso le varie Servlet, che a loro volta gestiranno la richiesta.

# Subsystem services

Infine, illustriamo i servizi esposti dai sottosistemi.

# Presentation Layer

Home

|  |  |  |
| --- | --- | --- |
| Catalogo | visualizzaProdotti | Permette a chi accede al sito di visionare i prodotti presenti nel catalogo. |

Acquisti

|  |  |  |
| --- | --- | --- |
| Ordine | ordiniUtente | Permette all’utente di visualizzare gli ordini  effettuati dallo stesso sul sito. |
| ordiniVenditore | Permette al venditore di  visualizzare tutti gli ordini effettuati sul sito. |

# Application Layer

GestioneAccount

|  |  |  |
| --- | --- | --- |
| Registrazione | Register | Consente agli utenti non registrati di effettuare la registrazione. |
| Autenticazione | Login | Consente agli utenti di  effettuare il login una volta inserite le proprie credenziali. |
| Logout | Consente agli utenti loggati di uscire dal proprio  account. |

GestioneProdotti

|  |  |  |
| --- | --- | --- |
| Ricerca | ricercaProdotto | Consente all’utente di cercare un prodotto specifico presente nel catalogo. |
| Catalogo | aggiungiProdotto | Consente ai venditori di  aggiungere nuovi prodotti al sito. |
| modificaProdotto | Consente ai venditori di modificare i dettagli di un  prodotto. |
| rimuoviProdotto | Consente ai venditori di  rimuovere dei prodotti dal sito. |

GestioneAcquisti

|  |  |  |
| --- | --- | --- |
| Acquisto | effettuaAcquisto | Consente all’utente di effettuare l’acquisto. |
| Wishlist | aggiungiWishlist | Consente all’utente di  aggiungere un prodotto alla propria wishlist. |
| rimuoveDaWishlist | Consente all’utente di  rimuovere un prodotto dalla propria wishlist. |
| Carrello | addToCart | Consente all’utente di  aggiungere un prodotto al carrello. |
| removeFromCart | Consente all’utente di  rimuovere un prodotto dal carrello. |
| modificaQuantità | Consente all’utente di  modificare la quantità di un prodotto presente all’interno del carrello. |

# Data management layer

DAO

|  |  |  |
| --- | --- | --- |
| Utente | retrieveUtenti | Permette di recuperare gli utenti dal database. |
| saveUtente | Permette di salvare un nuovo utente nel database |
| removeUtente | Permette di rimuovere un utente dal database |
| Ordine | retrieveOrdine | Permette di recuperare un ordine specifico |
| saveOrdine | Permette di salvare un ordine nel database |
| removeOrdine | Permette di eliminare un ordine nel database |
| RigaOrdine | retrieveRigaOrdine | Permette di recuperare uno specifico prodotto (appunto la riga) appartenente ad un |

|  |  |  |
| --- | --- | --- |
|  |  | ordine |
| saveRigaOrdine | Permette di salvare i prodotti (le righe)  dell’ordine appena effettuato |
| removeRigaOrdine | Permette di rimuovere la  riga selezionata dal database |
| Prodotto | retrieveProdotto | Permette di recuperare un prodotto specifico |
| saveProdotto | Permette di salvare un prodotto nel database |
| removeProdotto | Permette di eliminare un prodotto nel database |
| Wishlist | retrieveWishlist | Permette di recuperare una wishlist specifica |
| SaveWishlist | Permette di salvare una wishlist nel database |
| removeWishlist | Permette di eliminare una wishlist nel database |
| ItemWishlist | retrieveItemWishlist | Permette di recuperare un prodotto specifico da una wishlist |
| saveItemWishlist | Permette di salvare un prodotto in una wishlist |
| removeItemWishlist | Permette di eliminare un prodotto in una wishlist |