UNIVERSITÀ DEGLI STUDI DI SALERNO

Corso di Ingegneria del Software

A.A. 2023/2024

SecondLifeTech

OBJECT DESIGN DOCUMENT

VERSIONE 1.1

09/03/2024

DOCENTE:

PROF. ANDREA DE LUCIA

PRESENTATO DA:

CAPRICANO DAVIDE, 05121 10422

GIARDINETTO GIUSEPPE, 05121 14655

SPERA ALFREDO JESHOUA, 05121 12240 (PROJECT MANAGER)

REVISION HISTORY

Data	Versione	Descrizione
10/02/2024	0.1	Prima versione
12/02/2024	0.2	Scrittura packages
16/02/2024	0.3	Diagrammi
23/02/2024	1.0	Mapping dei path
09/03/2024	1.1	Correzioni

Sommario

1. Introduzione

1.1. Trade-off dell'Object Design

1.1.1. Tempo di Risposta vs Spazio Richiesto

Data la natura competitiva del mercato del dominio di applicazione, risulta molto più importante riuscire a garantire un servizio con un tempo di risposta che renda la navigazione del sito piacevole. In questo modo l'azienda sarà in grado di competere con altre dello stesso settore.

1.2. Linee Guida

1.2.1. Nominazione

Queste sono delle linee guida riguardanti la nominazione delle varie componenti del progetto.

Tutti i nomi utilizzati saranno in Inglese.

I seguenti componenti devono seguire la convenzione UpperCamelCase per il loro nome:

- Classi
- Interfacce
- Metodi
- Variabili

I nomi delle classi (per convenzione di Java) ed ogni parola all'interno del nome devono cominciare con una lettera maiuscola.

I nomi delle variabili e dei metodi seguono la camelCase.

I nomi dei packages dovranno essere in minuscolo. Se sono presenti più parole all'interno del nome verranno separate da un underscore ("_").

I nomi dei file HTML devono essere in minuscolo. Se sono presenti più parole all'interno del nome verranno separate da un trattino ("-").

1.2.2. Indentazione all'interno dei file

Un'indentazione consiste in una tabulazione, che è composta da 4 spazi.

Le parentesi di apertura del blocco di codice devono essere poste sulla stessa riga della firma del metodo separate da uno spazio dal blocco dei parametri

Le parentesi di chiusura del blocco di codice **non vuoto** devono essere correttamente indentate in corrispondenza del blocco a cui fanno da terminatori, inoltre devono essere poste una riga sotto l'ultima riga di codice del blocco.

Nel caso il blocco di codice sia vuoto la parentesi di chiusura deve trovarsi sulla stessa riga della parentesi di apertura.

L'esempio seguente riassume le indentazioni necessarie per il codice:

```
public CostruttoreClasse {}

public int metodo(int nomeParametro) {
    …
```

```
}
```

I file HTML devono possedere un'indentazione che rispetta la gerarchia dei tag.

1.2.3. Javadoc

Al di sopra del nome di ogni metodo, classe e interfaccia Java deve essere presente il Javadoc.

Questo deve contenere una breve descrizione, i parametri utilizzati e cosa ritorna. Esempio:

```
/**
 * Descrizione Classe
 */
public class NomeClasse {
    /**
     * Descrizione Metodo
     * @param nomeParametro Descrizione parametro
     * @return Descrizione ritorno
     */
    public void metodo(int nomeParametro) {
        …
    }
}
```

Non si deve scrivere il Javadoc al di sopra di un metodo che si trova all'interno di una interfaccia, perché è già scritto sul metodo che lo implementa.

1.3. Componenti off-the-shelf

In questo progetto si utilizzeranno diverse tecnologie:

- HTML, JS, CSS
- Database MySql
- Bootstrap 5.3.2 (componenti CSS)
- Moduli Spring

All'interno Spring è possibile usare dei moduli specifici sempre attraverso la Maven Central Repository.

Nome Modulo	Descrizione
`spring-boot-starter-web`	Starter per la creazione di applicazioni web, comprese quelle RESTful, utilizzando Spring MVC. Utilizza Tomcat come contenitore incorporato predefinito.
`spring-boot-starter-test`	Starter per testare le applicazioni Spring Boot con librerie quali JUnit Jupiter, Hamcrest e Mockito.
`json-path (2.9.0)`	Override della dipendenza con patch per la vulnerabilità CVE-2023-51074 di spring-boot-starter-test.

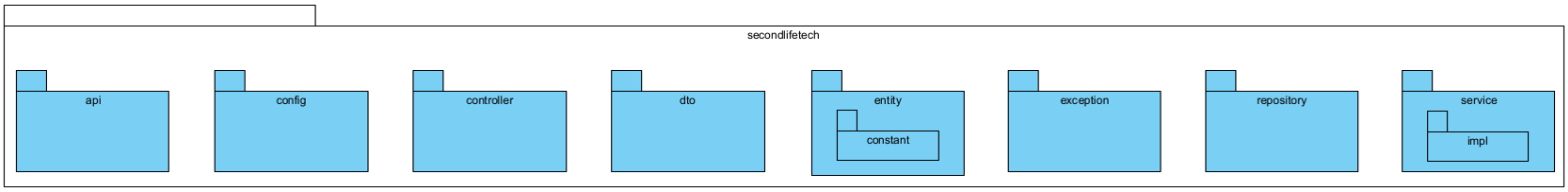
`spring-boot-starter-thymeleaf`	Template engine HTML per la creazione di applicazioni web MVC.
`lombok`	Annotazioni per semplificare lo sviluppo di Java automatizzando la generazione di codice boilerplate.
`spring-boot-starter-data-jpa`	Per implementare repository basate sulla Java Persistence API.
`mysql-connector-j`	Driver JDBC per connettersi a MySql.
`spring-boot-starter-security`	Framework di autenticazione e controllo degli accessi. Versione 6.
`spring-security-test`	Testing per Spring Security.
`thymeleaf-extras-springsecurity6`	Estensione di Thymeleaf per Spring Security 6.

1.4. Riferimenti

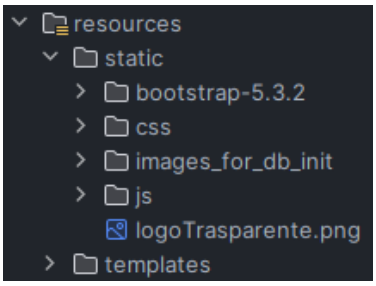
Questo documento si basa sul **Requirement Analysis Document** e sul **System Design Document**.

2. Divisione dei Packages

Il sistema, nel codice, sarà diviso nei seguenti package:



Inoltre, il sistema presenta una cartella Ýʃ Ê° ‡£Ýʃ all'interno di `resources` che contiene file HTML da utilizzare con Thymeleaf. La cartella `static` può contenere file CSS e JS.



2.1. Package “api”

Questo package contiene dei REST Controller:

- **`ImageRestController`**: responsabile per l’ottenimento delle immagini dei prodotti
- **`VariationsRestController`**: responsabile per l’ottenimento di un JSON utilizzato dalla pagina dei dettagli di un prodotto

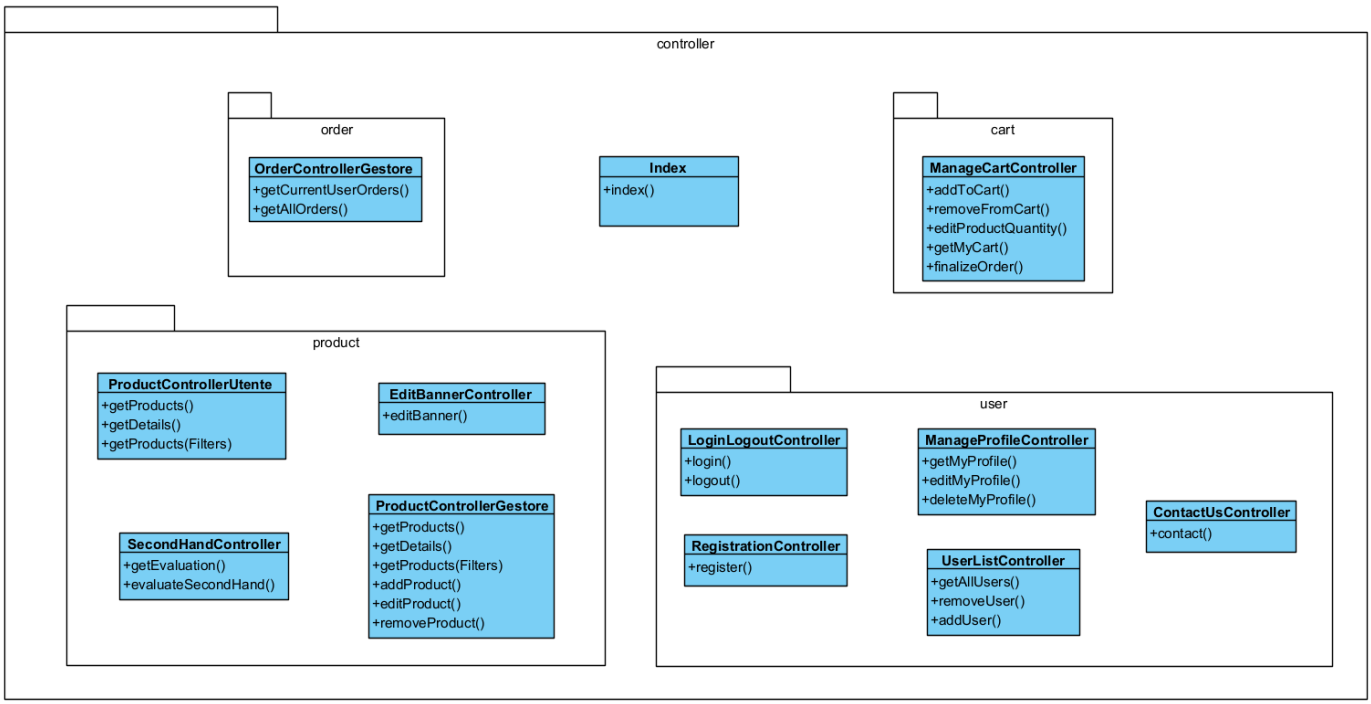
2.2. Package “config”

Questo package contiene classi Java che definiscono configurazioni per l'applicazione Spring Boot.

- **`CustomUserDetailsService`**: implementazione personalizzata dell'interfaccia `UserDetailsService` fornita da Spring Security. È responsabile di caricare i dettagli dell'utente (come nome utente, password e autorità) utilizzati dall'applicazione per l'autenticazione e l'autorizzazione.
- **`CustomAuthenticationSuccessHandler`**: implementazione personalizzata dell'interfaccia `AuthenticationSuccessHandler` fornita da Spring Security. È responsabile di caricare dai cookie del carrello gli oggetti e di inserirli all’interno del carrello nel database.
- **`DatabasePopulator`**: utilizzato per popolare il database all’inizio dell’applicazione.
- **`StartupRunner`**: un componente Spring Boot che implementa l'interfaccia `ApplicationRunner`. È progettata per eseguire operazioni di post-avvio dell'applicazione.
- **`WebSecurityConfig`**: una configurazione di Spring Security per gestire l'autenticazione degli utenti, l'autorizzazione alle risorse su pagine specifiche e le sessioni.

2.3. Package “controller”

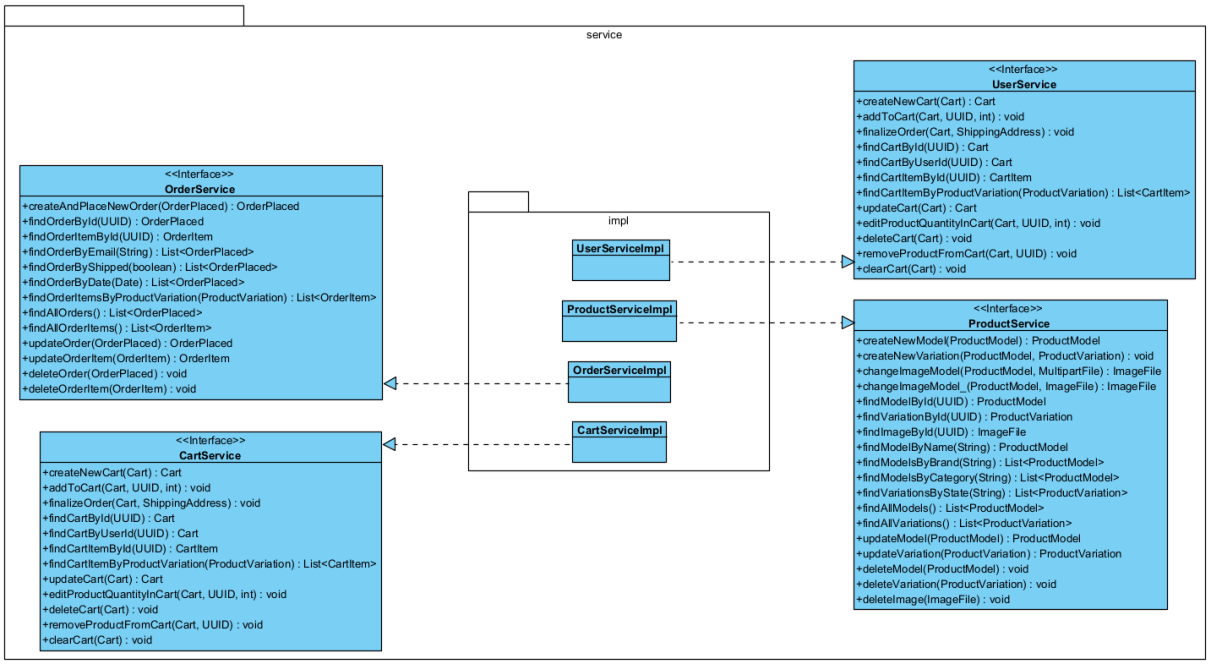
Questo package contiene le classi Java che fungono da punti di ingresso per le richieste HTTP nell'applicazione. Queste classi sono responsabili di accettare le richieste degli utenti provenienti dal web o da altre fonti e di elaborarle, spesso facendo riferimento ai servizi e alle entità dell'applicazione per soddisfare la richiesta.



2.4. Package “service”

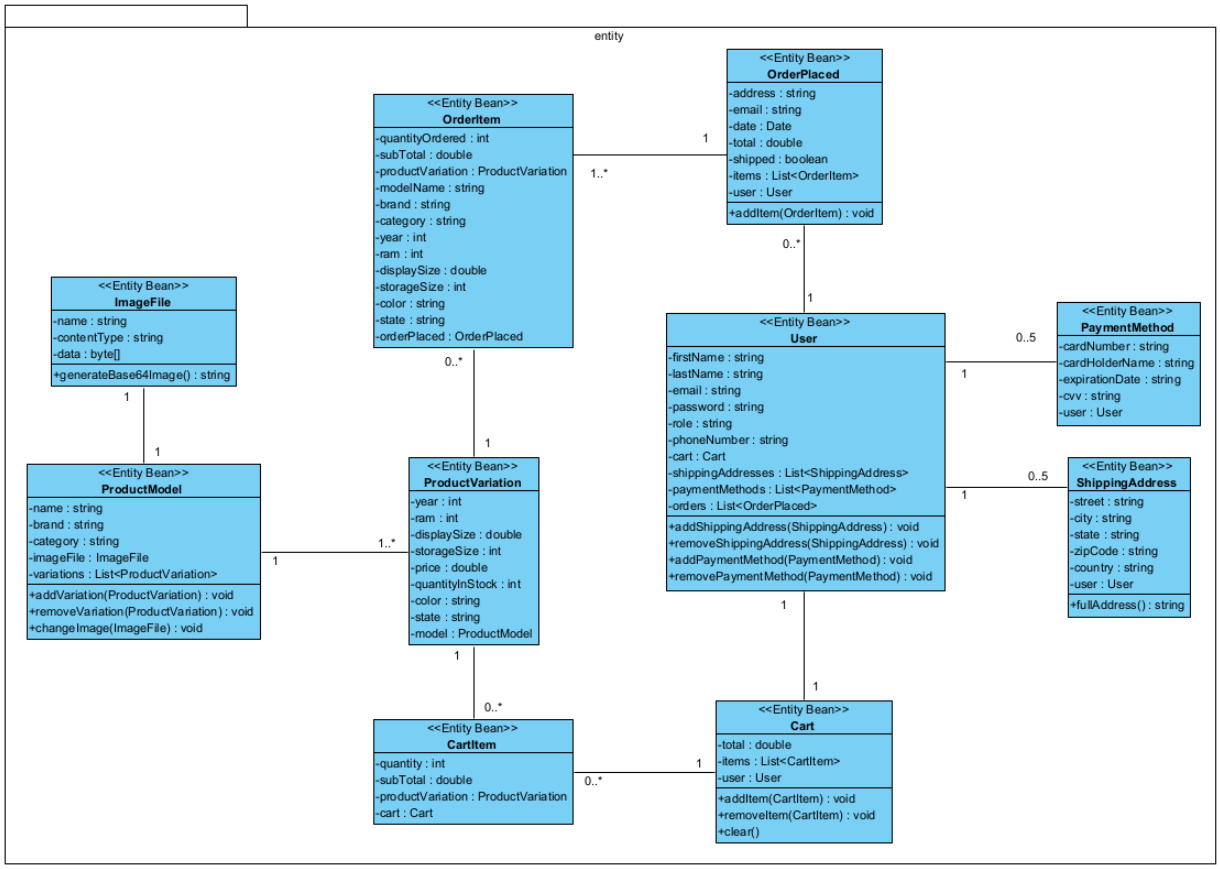
Questo package rappresenta il livello di applicazione. Contiene le classi Java che implementano la logica di business dell'applicazione.

Per utilizzare un servizio, è necessario chiamare la sua interfaccia. Spring eseguirà l’iniezione delle dipendenze per trovare l’implementazione adatta.



2.5. Package “entity”

Questo package contiene le classi Java che rappresentano le entità di dominio dell'applicazione.

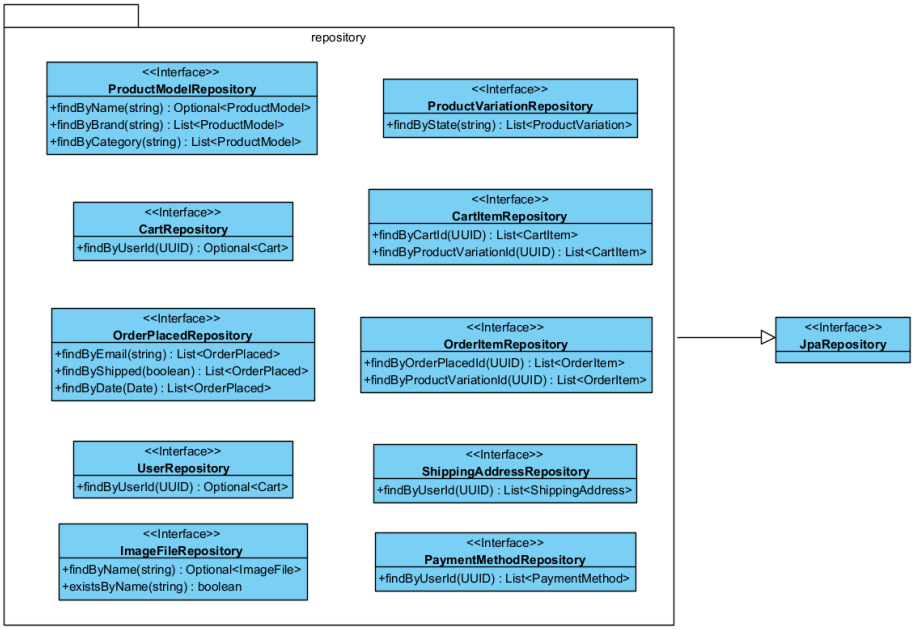


Contiene anche un package chiamato “**constant**”, utilizzato per contenere le stringhe costanti:

- **`ProductCategory`**: SMARTPHONE, TABLET
- **`ProductState`**: BUONO, OTTIMO, ACCETTABILE
- **`UserRole`**: ROLE_CLIENTE, ROLE_GESTORE_UTENTI, ROLE_GESTORE_PRODOTTI, ROLE_GESTORE_ORDINI
    - Sono preceduti da “ROLE_” per convenzione di Spring Security

2.6. Package “repository”

Questo package contiene le classi Java che gestiscono l'accesso e la persistenza dei dati nel database dell'applicazione.



2.7. Package “exception”

Questo package contiene tutte le eccezioni personalizzate che vengono lanciate dal sistema. Vengono lanciate principalmente dai servizi e risalgono verso i controller, dove devono essere gestite.

2.8. Package “dto”

Questo package contiene tutti i Data Transfer Object (dto) utilizzati per lo scambio di informazioni tra il presentation layer e l’application layer.

È utilizzata per la ricerca con filtri di prodotti, ordini e utenti, ma è anche usata dalla valutazione usato.

3. Mapping dei Path

Questi path derivano dai mock-ups (RAD 4.5 Mock-ups).

Path	Utenti autorizz.	Metodo Get	Metodi Post
/	Tutti	Homepage del sito	
/second-hand	Guest, Cliente	Pagina di valutazione dell'usato	Mandare i dati dell'usato
/products*?filters*	Guest, Cliente	Pagina di visualizzazione dei modelli in vendita *Filters*: filtri di ricerca	
/products/details/*{nome-modello}*	Guest, Cliente	Pagina di visualizzazione dei dettagli di un modello *Nome-modello*: nome del modello da visualizzare	
/register	Guest	Pagina di registrazione	
/register/save	Guest		Mandare i dati per la registrazione
/login	Guest	Pagina di login	Mandare i dati per il login
/logout	Utente Registrato	Effettua il logout (gestito da spring security)	Effettua il logout (gestito da spring security)
/my-cart	Cliente, Guest	Pagina per visualizzare il proprio carrello	
/my-cart/add	Cliente, Guest		Aggiunge una variazione al carrello
/my-cart/remove	Cliente, Guest		Rimuove una variazione al carrello
/my-cart/edit-quantity	Cliente, Guest		Modifica la quantità di una variazione nel carrello
/my-cart/clear	Cliente, Guest		Rimuove tutte le variazioni dal carrello
/my-profile	Cliente	Pagina per visualizzare il proprio profilo cliente	
/my-profile/edit-info	Cliente		Modifica i dati del cliente

/my-profile/edit-password	Cliente		Modifica la password del cliente
/my-profile/delete-my-account	Cliente		Elimina il profile dell'utente
/my-profile/shipping-addresses	Cliente	Pagina per visualizzare i propri indirizzi di spedizione	
/my-profile/shipping-addresses/add	Cliente		Aggiunge un indirizzo di spedizione
/my-profile/shipping-addresses/remove	Cliente		Rimuove un indirizzo di spedizione
/my-profile/shipping-addresses/edit	Cliente		Modifica un indirizzo di spedizione
/my-profile/payment-methods	Cliente	Pagina per visualizzare i propri metodi di pagamento	
/my-profile/payment-methods/add	Cliente		Aggiunge un metodo di pagamento
/my-profile/payment-methods/remove	Cliente		Rimuove un metodo di pagamento
/my-profile/orders?filters	Cliente	Pagina per visualizzare i propri ordini effettuati *Filters*: filtri di ricerca	
/my-cart/place-order	Cliente	Pagina per selezionare i dati per finalizzare un ordine	
/place-order/payment-in-progress*?id*	Cliente	Mock di una pagina di caricamento di una banca per il pagamento dell'ordine *Id*: identificatore dell'ordine	
/place-order/successful	Cliente	Pagina di successo per la finalizzazione dell'ordine	
/dashboard-prodotti	Gestore Prodotti	Pagina iniziale per i gestori prodotti	
/dashboard-prodotti/models*?filters*	Gestore Prodotti	Pagina per visualizzare tutti i modelli in vendita *Filters*: filtri di ricerca	
/dashboard-prodotti/models/add-model	Gestore Prodotti		Aggiunge un nuovo modello con i dati ottenuti

/dashboard-prodotti/models/remove-model	Gestore Prodotti		Rimuove il modello selezionato
/dashboard-prodotti/models/edit-model	Gestore Prodotti		Modifica il modello selezionato con i dati ottenuti
/dashboard-prodotti/models/*{nome-modello}***/variations**	Gestore Prodotti	Pagina per visualizzare tutte le variazioni di un modello	
/dashboard-prodotti/models/*{nome-modello}***/variations/add-variation**	Gestore Prodotti		Aggiunge una nuova variazione con i dati ottenuti
/dashboard-prodotti/models/*{nome-modello}***/variations/remove-variation**	Gestore Prodotti		Rimuove la variazione selezionata
/dashboard-prodotti/models/*{nome-modello}***/variations/edit-variation**	Gestore Prodotti		Modifica la variazione selezionata con i dati ottenuti
/order-admin/orders*?filters*	Gestore Ordini	Pagina principale del gestore ordini per visualizzare tutti gli ordini effettuati dagli utenti *Filters*: filtri di ricerca	Modifica lo stato di spedizione di un ordine specifico
/user-admin/users*?filters*	Gestore Utenti	Pagina principale del gestore utenti per visualizzare tutti gli utenti registrati al sistema *Filters*: filtri di ricerca	
/user-admin/users/add-user	Gestore Utenti	Pagina per inserire dati per un nuovo utente	Aggiunge un nuovo utente
/user-admin/users/remove-user	Gestore Utenti		Rimuove l'utente selezionato

4. Design Pattern

4.1. DAO Design Pattern

Per la persistenza, la manipolazione e la visualizzazione dei dati nel Database è stato impiegato il **Design Pattern DAO** (Data Access Object).

Questo pattern permette di separare le API di accesso ai dati dalla logica di business. L'implementazione del pattern è fornita dal modulo **Spring Data JPA**.

4.2. Façade Pattern

Per l'implementazione dei servizi dei sottosistemi dell'application layer è stato impiegato il Façade Pattern.

Questo pattern permette di accedere a servizi offerti dai sottosistemi tramite interfacce esposte che nascondono implementazioni complesse.

Nel nostro caso, ogni sottosistema possiede un'interfaccia chiamata *Service*, con un'opportuna classe chiamata *ServiceImpl* che implementa i metodi che corrispondono ai servizi offerti dal sottosistema.

Questi metodi a loro volta utilizzeranno le classi entity e le classi repository. L'output prodotto da queste classi sarà utilizzato poi dai Controller (o più in generale dal layer di presentazione).

Questo permette quindi una separazione netta tra la logica di business delegata ai *service* e alla presentazione delegata ai Controller.

5. Glossario

Termine	Definizione
Guest	Ospite
Browser	Software per la navigazione in Internet
Prodotto	Merce in vendita sul sito
Dispositivo	Sinonimo di “Prodotto”, può essere uno Smartphone o un Tablet
Modello	Particolare famiglia di prodotti
Variazione	Specializzazione di un modello
Second Hand	Usato
DB	Database
Dominio di Applicazione	Il settore in cui il sistema opera
Logica di Business	Le regole, procedure e le operazioni che definiscono il comportamento del sistema
Cookie	Memorizzazione dati sul browser dell’utente
Operazioni CRUD	Operazioni principali per interagire con le entità persistenti: Create (Crea), Read (Lettura), Update (Modifica), Delete (Elimina)
Web Server	Server dove viene eseguito il codice
DAO	Data Access Object
Principal	Utente autenticato nel sistema