

**Object Design Document**

[Versione 1]

**Sommario**

[**Informazioni sul documento** 2](#_Toc160292866)

[**Generalità** 2](#_Toc160292867)

[**Team Project** 2](#_Toc160292868)

[**Revision History** 3](#_Toc160292869)

[**1. Introduzione** 4](#_Toc160292870)

[**1.1 Object design trade-offs** 5](#_Toc160292871)

[**1.2 Linee guida per la documentazione dell’interfaccia** 6](#_Toc160292872)

[**1.3 Definizioni, acronimi e abbreviazioni** 6](#_Toc160292873)

[**1.4 Riferimenti** 6](#_Toc160292874)

[**2. Packages** 6](#_Toc160292875)

[**3. Class interfaces** 6](#_Toc160292876)

[**4. Design patterns** 22](#_Toc160292877)

[**5. Glossario** 24](#_Toc160292878)

**Informazioni sul documento**

**Generalità**

* **Progetto**: TechHeaven
* **Versione**: [Versione 1]
* **Documento**: Documento di object design
* **Data**: [26/02/2024]

**Team Project**

|  |  |  |  |
| --- | --- | --- | --- |
| Nome Membro | Matricola | Ruolo | Contatti |
| Dorotea Serrelli | 0512113740 | Project manager | d.serrelli1@studenti.unisa.it |
| Raffaella Sabatino | 0512115114 | Team member | r.sabatino17@studenti.unisa.it |

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| Data | Versione | Descrizione | Autore |
| 26/02/2024 | 0.1 | Stesura dell’introduzione al documento ODD e definizione dei trade-off e dei object design goals | Dorotea Serrelli |

**1. Introduzione**

La società TechHeavenSrl è responsabile dall’anno 2000 della gestione del negozio “TechHeaven – Il paradiso digitale”, specializzato nella vendita di prodotti elettronici, elettrodomestici, telefonia.

Tale negozio è, attualmente, un punto vendita di riferimento nella zona per lo smercio, la qualità e il prezzo dei prodotti, al punto che soddisfa un grande bacino di utenza, quasi esclusivamente residente nella provincia.

La società intende espandere i confini della propria attività ed ampliare la clientela, avvalendosi di un sistema software che consenta, sotto il profilo soggettivo, una maggiore conoscibilità della società e dell’affidabilità della stessa; sotto il profilo oggettivo, favorisca l’incremento della vendita dei prodotti.

Il sistema software verrà sviluppato per fornire alla clientela informazioni sulla società, sul punto vendita e sui prodotti trattati.

La piattaforma permetterà, infatti, al cliente di registrarsi, in modo da poter visionare i prodotti in vendita ed acquistarli, tenere traccia dello stato degli ordini effettuati presso il negozio online, creare una lista di prodotti desiderati (wishlist).

La piattaforma, inoltre, consentirà l’accesso ai seguenti dipendenti:

* Gestore degli ordini: responsabile del processo di acquisizione, registrazione ed evasione degli ordini dei clienti, nonché dell’elaborazione di richieste di approvvigionamento di prodotti da inoltrare, poi, all’ufficio acquisti.
* Gestore del catalogo: responsabile della presentazione, organizzazione e gestione del catalogo dei prodotti venduti dal negozio.

La piattaforma, quindi, consentirà al gestore degli ordini di visionare gli ordini commissionati dai clienti al negozio e gli ordini che sono stati spediti, preparare un ordine alla spedizione e fare richiesta di approvvigionamento di prodotti mancanti.

Essa, inoltre, permetterà al gestore del catalogo di visionare il catalogo e di poter inserire, cancellare e modificare un prodotto nel catalogo.

In questo documento verranno descritti i compromessi di progettazione degli oggetti effettuati, le linee guida seguite per le interfacce dei sottosistemi - riguardanti la nomenclatura, la documentazione e le convenzioni sui formati -, la decomposizione dei sottosistemi in packages e classi e le interfacce delle classi.

**1.1 Object design trade-offs**

Nella fase di progettazione degli oggetti del sistema si sono analizzati i seguenti trade-offs:

* **Spazio di memoria vs Tempo di risposta**

Maggiore spazio di memoria significa archiviare più dati, come la cronologia degli ordini, i dettagli dei prodotti e le informazioni sui clienti.

Privilegiare questo aspetto migliorerebbe l'esperienza utente e la funzionalità del software, ma può anche aumentare i costi e la complessità.

Un tempo di risposta rapido è fondamentale per un'esperienza utente fluida e per l'efficienza dell’esecuzione delle funzionalità offerte dal sistema. L'ottimizzazione del tempo di risposta può richiedere la memorizzazione nella cache di dati e l'utilizzo di algoritmi efficienti.

Visto che un tempo di risposta rapido è cruciale per la soddisfazione del cliente, la gestione fluida del processo di evasione degli ordini e la competitività del negozio online, si intende privilegiare il tempo di risposta.

* **Leggibilità vs Costi**

Un aspetto importante da prendere in considerazione nel nostro sistema è la leggibilità del codice: è da tenere conto che non necessariamente coloro che faranno manutenzione o monitoraggio del sistema saranno i creatori del sistema stesso.

Pertanto, si vorrà garantire la leggibilità del codice utilizzando commenti e documentazione dei vincoli di implementazione delle interfacce e classi coinvolte nel sistema, anche se ciò comporterà un aumento dei costi e del tempo di sviluppo.

* **Sicurezza vs Prestazioni**

Visto che il cliente sottolinea la necessità dello sviluppo di meccanismi di protezione agli attacchi informatici SQLInjection e Cross-site scripting e di integrità e riservatezza dei dati scambiati tra client e server, si intende privilegiare il requisito di sicurezza, a discapito del livello di prestazioni elevate che potenzialmente raggiungerebbe il sistema.

Tale decisione farà in modo che verrà garantito il livello di prestazioni nel tempo di risposta delineato nel primo trade-off discusso in questo paragrafo.

A seguire i trade-offs, si intende raggiungere i seguenti obiettivi:

* **Robustezza** : il sistema deve reagire correttamente a situazioni impreviste, attraverso il controllo degli errori e la gestione delle eccezioni.
* **Incapsulamento** : si vogliono nascondere i dettagli implementativi delle classi grazie all’utilizzo delle interfacce, rendendo possibile l’utilizzo di funzionalità offerte da diversi componenti o layer sottoforma di black-box.

**1.2 Linee guida per la documentazione dell’interfaccia**

**1.3 Definizioni, acronimi e abbreviazioni**

**1.4 Riferimenti**

Di seguito una lista di riferimenti ad altri documenti utili durante la lettura:

* Problem Statement
* RAD
* SDD
* Database Design Document

**2. Packages**

In questa sezione si illustra la suddivisione del sistema software in packages, in base a quanto definito nel documento di System Design.

Tale suddivisione è motivata dalle scelte architetturali definite nel SDD e ricalca la struttura di directory standard definita da Maven.

…

**3. Class interfaces**

In questa sezione sono illustrate le interfacce di ciascun package, eccetto il package Storage e le classi control.

**Package Registrazione**

|  |  |
| --- | --- |
| Nome classe | RegistrazioneService |
| Descrizione | Si occupa di gestire la registrazione dei nuovi clienti del negozio. |
| Metodi | +registraCliente(username: String, password: String, nome: String, cognome: String, sesso: Sex, email: String, telefono: String, indirizzoSped: Indirizzo) : Utente |
| Invariante di classe | Nessuno |
| Nome metodo | +registraCliente(username: String, password: String, nome: String, cognome: String, sesso: Sex, email: String, telefono: String, indirizzoSped: Indirizzo) |
| Descrizione | Il metodo permette di registrare un nuovo utente nel sistema (con ruolo Cliente) con le seguenti informazioni : username, password, nome, cognome, sesso, email, numero di telefono ed indirizzo di spedizione. |
| Precondizione | **context** RegistrazioneService:: registraCliente(username: String, password: String, nome: String, cognome: String, sesso: Sex, email: String, telefono: String, indirizzoSped: Indirizzo) **pre**:  Utente.checkValidate(username: String, password: String, nome: String, cognome: String, sesso: Sex, email: String, telefono: String, indirizzoSped: Indirizzo) |
| Postcondizione | **context** RegistrazioneService::registraCliente(username: String, password: String, nome: String, cognome: String, sesso: Sex, email: String, telefono: String, indirizzoSped: Indirizzo) **post**:  Utente.ifExists(username) |

**Package Autenticazione**

|  |  |
| --- | --- |
| Nome classe | AutenticazioneService |
| Descrizione | Si occupa di gestire l’autenticazione al sistema ad unico fattore del cliente, del gestore degli ordini e del gestore del catalogo; inoltre, si occupa delle funzionalità relative alla gestione dell’area riservata, delle informazioni personali del cliente, della reimpostazione della password e del cambio di ruolo del gestore del catalogo e del gestore degli ordini. |
| Metodi | +login(String username, String password) : Utente  +loginSelectedRole(String username, String password, String ruoloScelto) : Utente  +logout(Utente user) : void  +visualizzaAreaRiservata(Utente user) : boolean  +modificaDatiPersonali(Utente user, String infoSelected, String updatedData) : Utente  +inserimentoIndirizzoPersonale(Utente user, Indirizzo address) : Utente  +eliminazioneIndirizzoPersonale(Utente user, Indirizzo address) : Utente  +aggiornamentoIndirizzoPersonale(Utente user, Indirizzo address, Indirizzo newAddress) : Utente  +reimpostazionePassword(String username, String email, String newPassword) : boolean |
| Invariante di classe | Nessuno |
| Nome metodo | +login(String username, String password) : Utente |
| Descrizione | Il metodo permette di autenticare un utente nel sistema con un solo ruolo. |
| Precondizione | Nessuna |
| Postcondizione | **context** AutenticazioneService::login(username: String, password : String) **post**:  isLogged(username) |
| Nome metodo | +loginSelectedRole(String username, String password, String ruoloScelto) : Utente |
| Descrizione | Il metodo permette di autenticare un utente, che possiede più ruoli associati, nel sistema con il ruolo scelto. |
| Precondizione | Nessuna |
| Postcondizione | **context** AutenticazioneService::loginSelectedRole(username: String, password : String, ruoloScelto : String) **post**:  isLogged(username) |
| Nome metodo | +logout(Utente user) : void |
| Descrizione | Il metodo permette di disconnettere un utente dal sistema. |
| Precondizione | Nessuna |
| Postcondizione | **context** AutenticazioneService::logout(user : Utente) **post**:  !isLogged(user.getUsername()) |
| Nome metodo | +visualizzaAreaRiservata(Utente user) : boolean |
| Descrizione | Il metodo permette di visualizzare le informazioni personali dell’utente e gli ordini che ha effettuato. |
| Precondizione | **context** AutenticazioneService::visualizzaAreaRiservata(user : Utente) **pre**:  isLogged(user.getUsername()) |
| Postcondizione | nessuna |
| Nome metodo | +modificaDatiPersonali(Utente user, String infoSelected, String updatedData) : Utente |
| Descrizione | Il metodo permette di modificare il numero di telefono o l’indirizzo e-mail dell’utente. Il parametro *infoSelected* rappresenta l’informazione da modificare mentre il parametro *updatedData* indica la nuova informazione da memorizzare. |
| Precondizione | **context** AutenticazioneService:: modificaDatiPersonali(user: Utente, infoSelected: String, updatedData: String) **pre**:  isLogged(user.getUsername()) |
| Postcondizione | **context** AutenticazioneService:: modificaDatiPersonali(user:Utente, infoSelected : String, updatedData: String) **post**:  self.getEmail() == updatedData OR self.getTelefono() == updatedData |
| Nome metodo | +inserimentoIndirizzoPersonale(Utente user, Indirizzo address) : Utente |
| Descrizione | Il metodo effettua l’inserimento di un nuovo indirizzo di spedizione nella rubrica degli indirizzi personali dell’utente. |
| Precondizione | **context** AutenticazioneService:: inserimentoIndirizzoPersonale(user: Utente, address: Indirizzo) **pre**:  isLogged(user.getUsername()) AND !(user.getIndirizzi() -> includes(address)) |
| Postcondizione | **context** AutenticazioneService:: inserimentoIndirizzoPersonale(user: Utente, address: Indirizzo) **post**:  user.getIndirizzi() -> includes(address) |
| Nome metodo | +eliminazioneIndirizzoPersonale(Utente user, Indirizzo address) : Utente |
| Descrizione | Il metodo permette di eliminare un indirizzo di spedizione selezionato, *address*, dalla rubrica degli indirizzi dell’utente. |
| Precondizione | **context** AutenticazioneService:: eliminazioneIndirizzoPersonale(user: Utente, address: Indirizzo) **pre**:  isLogged(user.getUsername()) AND user.getIndirizzi() -> includes(address) |
| Postcondizione | **context** AutenticazioneService:: eliminazioneIndirizzoPersonale(user: Utente, address: Indirizzo) **post**:  !(user.getIndirizzi() -> includes(address)) |
| Nome metodo | +aggiornamentoIndirizzoPersonale(Utente user, Indirizzo address, Indirizzo newAddress) : Utente |
| Descrizione | Il metodo permette di aggiornare un indirizzo esistente, *address*, nella rubrica degli indirizzi dell’utente, memorizzando *newAddress*. |
| Precondizione | **context** AutenticazioneService:: aggiornamentoIndirizzoPersonale(user: Utente, address: Indirizzo, newAddress : Indirizzo) **pre**:  isLogged(user.getUsername()) AND user.getIndirizzi() -> includes(address) |
| Postcondizione | **context** AutenticazioneService:: aggiornamentoIndirizzoPersonale(user: Utente, address: Indirizzo, newAddress : Indirizzo) **post**:  !(user.getIndirizzi() -> includes(address)) AND user.getIndirizzi() -> includes(newAddress) |
| Nome metodo | +reimpostazionePassword(String username, String email, String newPassword) : boolean |
| Descrizione | Il metodo permette di reimpostare la password di un utente richiedendo le credenziali *username*  e *email*. La nuova password è rappresentata da *newPassword*. |
| Precondizione | **context** AutenticazioneService:: reimpostazionePassword(String username, String email, String newPassword) **pre**:  Utente.checkPassword(newPassword) |
| Postcondizione | **context** AutenticazioneService:: reimpostazionePassword(String username, String email) **post**:  (UtenteDAO. retrieveUserByUsername(username)).getPassword() == newPassword |

**Package Navigazione**

|  |  |
| --- | --- |
| Nome classe | NavigazioneService |
| Descrizione | Si occupa di gestire la visualizzazione delle specifiche di un prodotto e della ricerca dei prodotti mediante menu di navigazione e barra di ricerca, nonché della pagina dei risultati ottenuti dalla ricerca. |
| Metodi | +visualizzaProdotto(Prodotto prod) : void  +ricercaProdottoMenu(String category) : Collection(Prodotto)  +ricercaProdottoBar(String keyword) : Collection(Prodotto) |
| Invariante di classe | Nessuno |
| Nome metodo | +visualizzaProdotto(Prodotto prod) : void |
| Descrizione | Il metodo permette di visualizzare le specifiche di un prodotto *prod* selezionato dal cliente. |
| Precondizione | Nessuna |
| Postcondizione | Nessuna |
| Nome metodo | +ricercaProdottoMenu(String category) : Collection(Prodotto) |
| Descrizione | Il metodo restituisce l’insieme dei prodotti che appartengono alla categoria *category* selezionata. |
| Precondizione | Nessuna |
| Postcondizione | **context** NavigazioneService:: ricercaProdottoMenu(category: String) **post**:  result->forAll(element | element.getCategoria() == category) |
| Nome metodo | +ricercaProdottoBar(String keyword) : Collection(Prodotto) |
| Descrizione | Il metodo restituisce l’insieme dei prodotti che hanno nel proprio nome o nella propria descrizione (dettagliata o di presentazione) la parola *keyword*. |
| Precondizione | Nessuna |
| Postcondizione | **context** NavigazioneService:: ricercaProdottoBar(keyword: String) **post**:  result->((exists(element | element.getNome() == keyword)) OR (exists(element | element.getTopDescrizione() == keyword)) OR (exists(element | element.getDettagli() == keyword))) |

**Package GestioneCarrello**

|  |  |
| --- | --- |
| Nome classe | Carrello |
| Descrizione | Si occupa di gestire le operazioni relative alla gestione del carrello: visualizzazione, inserimento prodotti nel carrello, eliminazione prodotti dal carrello, aumento delle quantità di un prodotto del carrello, decremento delle quantità di un prodotto del carrello. |
| Metodi | +visualizzaCarrello(Carrello cart) : Set(ProdottoCarrello)  +aggiungiProdottoAlCarrello(Carrello cart, ProdottoCarrello prod) : Carrello  +rimuoviProdottoDalCarrello(Carrello cart, ProdottoCarrello prod) : Carrello  +aumentaQuantitàProdotto(Carrello cart, ProdottoCarrello prod, int quantity) : Carrello  +decrementaQuantitàProdotto(Carrello cart, ProdottoCarrello prod, int quantity) : Carrello |
| Invariante di classe | Nessuna |
| Nome metodo | +visualizzaCarrello(Carrello cart) : Set(ProdottoCarrello) |
| Descrizione | Il metodo fornisce i prodotti presenti nel carrello virtuale. |
| Precondizione | Nessuna |
| Postcondizione | Nessuna |
| Nome metodo | +aggiungiProdottoAlCarrello(Carrello cart, ProdottoCarrello prod) : Carrello |
| Descrizione | Il metodo aggiunge un prodotto *prod* (di quantità 1) al carrello *cart*. |
| Precondizione | **context** Carrello:: aggiungiProdottoAlCarrello(cart: Carrello, prod: ProdottoCarrello) **pre**:  !(cart.prodotti -> includes(prod)) |
| Postcondizione | **context** Carrello:: aggiungiProdottoAlCarrello(cart: Carrello, prod: ProdottoCarrello) **post**:  cart.prodotti -> includes(prod) |
| Nome metodo | +rimuoviProdottoDalCarrello(Carrello cart, ProdottoCarrello prod) : Carrello |
| Descrizione | Il metodo rimuove il prodotto *prod* dal carrello *cart*. |
| Precondizione | **context** Carrello:: rimuoviProdottoDalCarrello(cart: Carrello, prod: ProdottoCarrello) **pre**:  cart.prodotti -> includes(prod) |
| Postcondizione | **context** Carrello:: rimuoviProdottoDalCarrello(cart: Carrello, prod: ProdottoCarrello) **post**:  !(cart.prodotti -> includes(prod)) |
| Nome metodo | +aumentaQuantitàProdotto(Carrello cart, ProdottoCarrello prod, int quantity) : Carrello |
| Descrizione | Il metodo aumenta la quantità di un prodotto *prod* del carrello *cart*, impostandola a *quantity*. |
| Precondizione | **context** Carrello:: aumentaQuantitàProdotto(cart: Carrello, prod: ProdottoCarrello, quantity: int) **pre**:  cart.prodotti -> includes(prod) AND  (cart.prodotti->select(p | p = prod).getQuantità() < quantity) AND quantity > 0 |
| Postcondizione | **context** Carrello:: aumentaQuantitàProdotto(cart: Carrello, prod: ProdottoCarrello, quantity: int) **post**:  (cart.prodotti->select(p | p = prod).getQuantità() == quantity) |
| Nome metodo | +decrementaQuantitàProdotto(Carrello cart, ProdottoCarrello prod, int quantity) : Carrello |
| Descrizione | Il metodo diminuisce la quantità di un prodotto *prod* del carrello *cart*, impostandola a *quantity*. |
| Precondizione | **context** Carrello:: decrementaQuantitàProdotto(Carrello cart, ProdottoCarrello prod, int quantity) **pre**:  cart.prodotti -> includes(prod) AND  (cart.prodotti->select(p | p = prod).getQuantità() > quantity) AND quantity > 0 |
| Postcondizione | **context** Carrello:: decrementaQuantitàProdotto(Carrello cart, ProdottoCarrello prod, int quantity) **post**:  (cart.prodotti->select(p | p = prod).getQuantità() == quantity) |

**Package GestioneWishlist**

|  |  |
| --- | --- |
| Nome classe | Wishlist |
| Descrizione | Si occupa di gestire le operazioni relative alla gestione della wishlist: visualizzazione, inserimento prodotti nella wishlist, eliminazione prodotti dalla wishlist. |
| Metodi | +visualizzaWishlist(Wishlist wishes, Utente user) : Set(Prodotto)  +aggiungiProdottoInWishlist(Wishlist wishes, Prodotto prod, Utente user) : Wishlist  +rimuoviProdottoDaWishlist(Wishlist wishes, Prodotto prod, Utente user) : Wishlist |
| Invariante di classe | Nessuna |
| Nome metodo | + visualizzaWishlist(Wishlist wishes, Utente user) : Set(Prodotto) |
| Descrizione | Il metodo fornisce i prodotti presenti nella wishlist dell’utente. |
| Precondizione | **context** Wishlist:: visualizzaWishlist(wishes: Wishlist, user:Utente) **pre**:  isLogged(user.getUsername()) |
| Postcondizione | Nessuna |
| Nome metodo | +aggiungiProdottoInWishlist(Wishlist wishes, Prodotto prod, Utente user) : Wishlist |
| Descrizione | Il metodo aggiunge un prodotto *prod* nella wishlist *wishes*. |
| Precondizione | **context** Wishlist:: aggiungiProdottoInWishlist(wishes: Wishlist, prod: Prodotto, user:Utente) **pre**:  !(wishes.prodotti -> includes(prod)) AND isLogged(user.getUsername()) |
| Postcondizione | **context** Wishlist:: aggiungiProdottoInWishlist(wishes: Wishlist, prod: Prodotto, user:Utente) **post**:  wishes.prodotti -> includes(prod) |
| Nome metodo | +rimuoviProdottoDaWishlist(Wishlist wishes, Prodotto prod, Utente user) : Wishlist |
| Descrizione | Il metodo rimuove il prodotto *prod* dalla wishlist *wishes*. |
| Precondizione | **context** Wishlist:: rimuoviProdottoDaWishlist(wishes: Wishlist, prod: Prodotto, user:Utente) **pre**:  wishes.prodotti -> includes(prod) AND isLogged(user.getUsername()) |
| Postcondizione | **context** Wishlist:: rimuoviProdottoDaWishlist(wishes: Wishlist, prod: Prodotto, user:Utente) **post**:  !(wishes.prodotti -> includes(prod)) |

**Package GestioneOrdini**

|  |  |
| --- | --- |
| Nome classe | GestioneOrdiniService |
| Descrizione | Si occupa di gestire le operazioni relative alla gestione degli ordini: creazione di un ordine, visualizzazione ordini evasi, visualizzazione ordini commissionati da evadere, preparazione di un ordine alla spedizione. |
| Metodi | +visualizzaOrdiniEvasi(Utente user) : Collection(Ordine)  +visualizzaOrdiniDaEvadere(Utente user): Collection(Ordine)  +creazioneOrdine (Carrello cart, Utente user) : void  +preparazioneSpedizioneOrdine(Ordine order, Utente user) : void |
| Invariante di classe | Nessuna |
| Nome metodo | +visualizzaOrdiniEvasi(Utente user) : Collection(Ordine) |
| Descrizione | Il metodo fornisce gli ordini che sono stati commissionati al negozio online e che sono stati evasi correttamente. |
| Precondizione | **context** GestioneOrdiniService:: visualizzaOrdiniEvasi(user:Utente) **pre**:  isLoggedAsOrderManager(user) |
| Postcondizione | **context** GestioneOrdiniService :: visualizzaOrdiniEvasi(user:Utente) **post**:  result -> forAll(o : Ordine | o.getStato() == ‘Spedito’) |
| Nome metodo | +visualizzaOrdiniDaEvadere(Utente user): Collection(Ordine) |
| Descrizione | Il metodo fornisce gli ordini che sono stati commissionati al negozio online e che devono essere ancora spediti. |
| Precondizione | **context** GestioneOrdiniService:: visualizzaOrdiniDaEvadere(user:Utente) **pre**:  isLoggedAsOrderManager(user) |
| Postcondizione | **context** GestioneOrdiniService :: visualizzaOrdiniDaEvadere(user:Utente) **post**:  result -> forAll(o : Ordine | o.getStato() == ‘Richiesta effettuata’) OR forAll(o : Ordine | o.getStato() == ‘Preparazione incompleta’) |
| Nome metodo | +creazioneOrdine (Carrello cart, Utente user) : void |
| Descrizione | Il metodo permette l’acquisto dei prodotti nel carrello *cart* da parte dell’utente *user*. |
| Precondizione | **context** GestioneOrdiniService:: creazioneOrdine (cart: Carrello, user: Utente) **pre**:  isLogged(user.getUsername()) AND  cart.prodotti -> size() > 0 |
| Postcondizione | **context** GestioneOrdiniService:: creazioneOrdine (cart: Carrello, user: Utente) **post**:  (cart.prodotti -> size() == 0) |
| Nome metodo | +preparazioneSpedizioneOrdine(Ordine order, Utente user) : void |
| Descrizione | Il metodo permette di preparare un ordine *order* alla spedizione. |
| Precondizione | **context** GestioneOrdiniService:: preparazioneSpedizioneOrdine(order:Ordine, user:Utente) **pre**:  (order.getStato() == ‘Richiesta effettuata’ OR order.getStato() == ‘Preparazione incompleta’) AND isLoggedAsOrderManager(user) |
| Postcondizione | **context** GestioneOrdiniService:: preparazioneSpedizioneOrdine(order:Ordine, user:Utente) **post**:  order.getStato() == ‘Spedito’ |

**Package GestioneApprovvigionamenti**

|  |  |
| --- | --- |
| Nome classe | GestioneApprovvigionamentiService |
| Descrizione | Si occupa di gestire le operazioni relative alla gestione delle richieste di approvvigionamento di prodotti: creazione di una richiesta di approvvigionamento e visualizzazione delle richieste di rifornimento effettuate. |
| Metodi | +visualizzaRichiesteFornitura(Utente user) : Collection(RichiestaApprovvigionamento)  +effettuaRichiestaApprovvigionamento(Prodotto prod, Utente user) : void |
| Invariante di classe | Nessuna |
| Nome metodo | +visualizzaRichiesteFornitura(Utente user) : Collection(RichiestaApprovvigionamento) |
| Descrizione | Il metodo fornisce le richieste di rifornimento effettuate dal negozio. |
| Precondizione | **context** GestioneApprovvigionamentiService:: visualizzaRichiesteFornitura(user:Utente) **pre**:  isLoggedAsOrderManager(user) |
| Postcondizione | **context** GestioneApprovvigionamentiService:: visualizzaRichiesteFornitura(user:Utente) **post**:  result -> forAll(o : Ordine | o.getStato() == ‘Spedito’) |
| Nome metodo | +effettuaRichiestaApprovvigionamento(Prodotto prod, Utente user) : void |
| Descrizione | Il metodo permette di creare una richiesta di approvvigionamento del prodotto *prod*. |
| Precondizione | **context** GestioneApprovvigionamentiService:: effettuaRichiestaApprovvigionamento(prod:Prodotto, user:Utente) **pre**:  isLoggedAsOrderManager(user) AND prod.getQuantitàDisponibile() == 0 |
| Postcondizione | Nessuna |

**Package Pagamento**

|  |  |
| --- | --- |
| Nome classe | PagamentoService |
| Descrizione | Si occupa di gestire le operazioni relative al pagamento degli ordini di un cliente: effettuare il pagamento e stampare un avviso di pagamento (ricevuta di pagamento). |
| Metodi | +stampaRicevuta(Utente user, Ordine order) : Pagamento  +effettuaPagamento(Utente user, Ordine order) : void |
| Invariante di classe | Nessuna |
| Nome metodo | +stampaRicevuta(Utente user, Ordine order) : Pagamento |
| Descrizione | Il metodostampa l’avviso di pagamento relativo all’ordine *order* effettuato dal cliente *user*. |
| Precondizione | **context** PagamentoService:: stampaRicevuta(user: Utente, order: Ordine) **pre**:  isLogged(user.getUsername()) |
| Postcondizione | Nessuna |
| Nome metodo | +effettuaPagamento(Utente user, Ordine order) : void |
| Descrizione | Il metodo effettua il pagamento dell’ordine *order* del cliente *user*. |
| Precondizione | **context** PagamentoService:: effettuaPagamento(user: Utente, order: Ordine) **pre**:  isLogged(user.getUsername()) |
| Postcondizione | Nessuna |

**Package GestioneCatalogo**

|  |  |
| --- | --- |
| Nome classe | GestioneCatalogoService |
| Descrizione | Si occupa di gestire le operazioni relative alla gestione del catalogo dei prodotti del negozio: visualizzazione del catalogo, inserimento di prodotti nel catalogo, eliminazione di prodotti dal catalogo ed aggiornamento delle specifiche di un prodotto del catalogo. |
| Metodi | +visualizzaCatalogo(Utente user) : Collection(Prodotto)  +aggiuntaProdottoInCatalogo(Utente user, Prodotto prod) : Collection(Prodotto)  +rimozioneProdottoDaCatalogo(Utente user, Prododtto prod) : Collection(Prodotto)  + aggiornamentoSpecificheProdotto(Utente user, Prodotto prod, String infoSelected, String updatedData) : Collection(Prodotto)  +aggiornamentoDisponibilitàProdotto(Utente user, Prodotto prod, int quantity) : Collection(Prodotto)  +aggiornamentoPrezzoProdotto(Utente user, Prodotto prod, float price) : Collection(Prodotto)  +inserimentoTopImage(Utente user, Prodotto prod, Immagine img) : Collection(Prodotto)  +inserimentoImmagineInGalleriaImmagini(Utente user, Prodotto prod, Immagine img) : Collection(Prodotto)  +cancellazioneImmagineInGalleria(Utente user, Prodotto prod, Immagine img) : Collection(Prodotto) |
| Invariante di classe | Nessuna |
| Nome metodo | +visualizzaCatalogo(Utente user) : Collection(Prodotto) |
| Descrizione | Il metodo fornisce il catalogo dei prodotti venduti dal negozio. |
| Precondizione | **context** GestioneCatalogoService:: visualizzaCatalogo(user: Utente) **pre**:  isLoggedAsCatalogueManager(user) |
| Postcondizione | **context** GestioneCatalogoService:: visualizzaCatalogo(user: Utente) **post**:  result -> forAll(o : Prodotto | o.getInCatalogo() == True) |
| Nome metodo | +aggiuntaProdottoInCatalogo(Utente user, Prodotto prod) : Collection(Prodotto) |
| Descrizione | Il metodo permette di aggiungere un prodotto *prod* nel catalogo. |
| Precondizione | **context** GestioneCatalogoService:: aggiuntaProdottoInCatalogo(user: Utente, prod: Prodotto) **pre**:  isLoggedAsCatalogueManager(user) |
| Postcondizione | **context** GestioneCatalogoService:: aggiuntaProdottoInCatalogo(user: Utente, prod: Prodotto) **post**:  result -> includes(prod) |
| Nome metodo | +rimozioneProdottoDaCatalogo(Utente user, Prodotto prod) : Collection(Prodotto) |
| Descrizione | Il metodo permette di cancellare un prodotto dal catalogo del negozio. |
| Precondizione | **context** GestioneCatalogoService:: rimozioneProdottoDaCatalogo(user: Utente, prod: Prodotto) **pre**:  isLoggedAsCatalogueManager(user) |
| Postcondizione | **context** GestioneCatalogoService:: rimozioneProdottoDaCatalogo(Utente user, Prodotto prod) **post**:  !(result -> includes(prod)) |
| Nome metodo | + aggiornamentoSpecificheProdotto(Utente user, Prodotto prod, String infoSelected, String updatedData) : Collection(Prodotto) |
| Descrizione | Il metodo permette di aggiornare le seguenti specifiche del prodotto: modello, marca, descrizione in evidenza, descrizione dettagliata, categoria, sottocategoria, messa in evidenza. Il parametro *infoSelected* è l’informazione che si vorrebbe modificare mentre *updatedData* è la nuova informazione da memorizzare. |
| Precondizione | **context** GestioneCatalogoService:: aggiornamentoSpecificheProdotto(user: Utente, prod:Prodotto, infoSelected: String, updatedData: String) **pre**:  isLoggedAsCatalogueManager(user) |
| Postcondizione | **context** GestioneCatalogoService:: aggiornamentoSpecificheProdotto(user: Utente, prod : Prodotto, infoSelected: String, updatedData: String) **post**:  (prod.getModello() == updatedData) OR (prod.getMarca() == updatedData) OR (prod.getTopDescrizione() == updatedData) OR (prod.getDettagli() == updatedData) OR (prod.getCategoria() == updatedData) OR (prod.getSottocategoria() == updatedData) OR (prod.getInEvidenza() == updatedData) |
| Nome metodo | +aggiornamentoDisponibilitàProdotto(Utente user, Prodotto prod, int quantity) : Collection(Prodotto) |
| Descrizione | Il metodo permette di aggiornare la quantità di scorte in magazzino del prodotto *prod*. |
| Precondizione | **context** GestioneCatalogoService:: aggiornamentoDisponibilitàProdotto(user: Utente, prod: Prodotto, quantity: int) **pre**:  isLoggedAsCatalogueManager(user) AND (quantity > prod.getQuantitàDisponibile() AND quantity > 0) |
| Postcondizione | **context** GestioneCatalogoService:: aggiornamentoDisponibilitàProdotto(user: Utente, prod: Prodotto, quantity: int) **post**:  prod.getQuantitàDisponibile() == quantity |
| Nome metodo | +aggiornamentoPrezzoProdotto(Utente user, Prodotto prod, int price) : Collection(Prodotto) |
| Descrizione | Il metodo permette di aggiornare il prezzo del prodotto *prod* a *price*. |
| Precondizione | **context** GestioneCatalogoService:: aggiornamentoPrezzoProdotto(user: Utente, prod: Prodotto, price: float) **pre**:  isLoggedAsCatalogueManager(user) AND price > 0.0 |
| Postcondizione | **context** GestioneCatalogoService:: aggiornamentoPrezzoProdotto(user: Utente, prod: Prodotto, price: float) **post**:  prod.getPrezzo() == price |
| Nome metodo | +inserimentoTopImage(Utente user, Prodotto prod, Immagine img) : Collection(Prodotto) |
| Descrizione | Il metodo permette di aggiungere l’immagine di presentazione *img* al prodotto *prod*. |
| Precondizione | **context** GestioneCatalogoService:: inserimentoTopImage(user: Utente, prod: Prodotto, img: Immagine) **pre**:  isLoggedAsCatalogueManager(user) AND prod.getTopImmagine() == null |
| Postcondizione | **context** GestioneCatalogoService:: inserimentoTopImage(user: Utente, prod: Prodotto, img: Immagine) **post**:  prod.getTopImmagine() == img |
| Nome metodo | +inserimentoImmagineInGalleriaImmagini(Utente user, Prodotto prod, Immagine img) : Collection(Prodotto) |
| Descrizione | Il metodo permette di aggiungere l’immagine di dettaglio *img* all’insieme delle immagini di dettaglio del prodotto *prod*. |
| Precondizione | **context** GestioneCatalogoService:: inserimentoImmagineInGalleriaImmagini(Utente user, Prodotto prod, Immagine img) **pre**:  isLoggedAsCatalogueManager(user) AND !(prod.getGalleriaImmagini() -> includes(img)) |
| Postcondizione | **context** GestioneCatalogoService:: inserimentoImmagineInGalleriaImmagini(Utente user, Prodotto prod, Immagine img) **post**:  prod.getGalleriaImmagini() -> includes(img) |
| Nome metodo | +cancellazioneImmagineInGalleria(Utente user, Prodotto prod, Immagine img) : Collection(Prodotto) |
| Descrizione | Il metodo permette di cancellare l’immagine di dettaglio *img* all’insieme delle immagini di dettaglio del prodotto *prod*. |
| Precondizione | **context** GestioneCatalogoService:: cancellazioneImmagineInGalleria(Utente user, Prodotto prod, Immagine img) **pre**:  isLoggedAsCatalogueManager(user) AND prod.getGalleriaImmagini() -> includes(img) |
| Postcondizione | **context** GestioneCatalogoService:: cancellazioneImmagineInGalleria(Utente user, Prodotto prod, Immagine img) **post**:  prod.getGalleriaImmagini() -> includes(img) |

**4. Design patterns**

In questo paragrafo si intende trattare dei design patterns scelti per garantire alcuni system design goals, scelti in base ai design objects trade-offs individuati.

Per ogni pattern utilizzato nello sviluppo dell’applicativo TechHeaven si darà:

* Una brevissima introduzione teorica.
* Il problema che doveva risolvere all’interno di TechHeaven.
* Una brevissima spiegazione della soluzione al problema individuato.
* Un grafico della struttura delle classi che implementano il pattern.

**Proxy design pattern**

Il Proxy design pattern pattern permette di controllare l'accesso ad un oggetto, fornendo un surrogato o un'interfaccia di sostituzione a questo oggetto.

Il ProxyObject agisce per conto di una classe RealObject: esso memorizza un sottoinsieme degli attributi del RealObject e gestisce completamente determinate richieste (ad esempio, determinare la dimensione di un'immagine), mentre altre richieste vengono delegate al RealObject.

Dopo la delega, viene creato il RealObject e caricato in memoria.

Questo design pattern è stato scelto tenendo conto del trade-off *Spazio di memoria vs Tempo di risposta* definito in questo documento: per migliorare le performance del sistema, si rimandano le computazioni più dispendiose (es. creazione di un oggetto on-demand, gestione di risorse remote, gestione degli accessi) ad un momento successivo (quando si ha bisogno effettivamente di quell’oggetto).

Visto che la visualizzazione delle immagini in evidenza e della galleria di immagini dettagliate dei prodotti ottenuti a seguito di una ricerca (per menù di navigazione o per barra di ricerca) è costosa, si intende caricare solo l’immagine in evidenza e le caratteristiche più rilevanti di un prodotto (nome, prezzo, brand, categoria).

Nel caso in cui l’utente volese approfondire le specifiche di un prodotto allora verrà caricato in memoria il prodotto reale, con la sua galleria di immagini e tutte le sue specifiche.

Tale design pattern viene adottato anche per il caricamento delle informazioni di un utente a seguito dell’autenticazione.

Analogamente, si intende tenere traccia delle informazioni anagrafiche dell’utente. Nel caso in cui l’utente volesse visionare lo storico degli ordini effettuato in negozio, accedendo all’area riservata, allora verranno caricati in memroia tutti gli ordini effettuati dall’utente.

… foto codice

**Facade design pattern**

Il Facade Design Pattern fornisce un'interfaccia unificata semplificata per un insieme di interfacce in un sottosistema più complesso, facilitando l'utilizzo di tali interfacce senza dover conoscere i dettagli interni.

Il design pattern è stato scelto perché risulta essere una soluzione efficace per garantire l'incapsulamento e gestire il trade-off tra leggibilità del codice e costi nel contesto descritto.

Esso rappresenta un modo per mantenere il codice pulito, semplice e facilmente comprensibile.

I vantaggi raggiunti da questo design pattern sono:

* nascondere al client i componenti di sistema, garantendo maggiore manutenibilità (viene invocato un solo oggetto);
* ridurre le dipendenze tra un sottoinsieme di sottosistemi ed il resto del sistema.

Essendo il nostro sistema molto complesso, si sfrutta il design pattern Facade per implementare tutta la sua logica di business e rendere più facile l’interfacciarsi con essa. Nello specifico, “TechHeaven” utilizza tale design pattern per ogni suo sottosistema, implementandolo attraverso delle interfacce che sono usate per accedere ai metodi interni.

**5. Glossario**