Progetto: PC Ready

Sommario

[**Ideazione e analisi dei requisiti** 5](#_Toc32937196)

[Requisiti 5](#_Toc32937197)

[Caso d’uso 1: Crea configurazione cliente 5](#_Toc32937198)

[Caso d’uso 2: Crea bundle 8](#_Toc32937199)

[Caso d’uso 3: Inserisci nuovo componente 11](#_Toc32937200)

[Caso d’uso 4: Inserisci copia componente 12](#_Toc32937201)

[Caso d’uso 5: Effettua acquisto 13](#_Toc32937202)

[Caso d’uso 6: Registrazione cliente 15](#_Toc32937203)

[Caso d’uso 7: Login 17](#_Toc32937204)

[Caso d’uso 8: Rimozione componente 18](#_Toc32937205)

[Caso d’uso 9: Crea promozione 19](#_Toc32937206)

[Caso d’uso 10: Modifica ordine 20](#_Toc32937207)

[Regole di dominio 21](#_Toc32937208)

[Glossario 22](#_Toc32937209)

[**Documento di visione** 24](#_Toc32937210)

[Introduzione 24](#_Toc32937211)

[Scopo 24](#_Toc32937212)

[Portata 24](#_Toc32937213)

[Definizioni, acronimi e abbreviazioni 25](#_Toc32937214)

[Posizionamento 25](#_Toc32937215)

[Parti interessate e descrizioni utente 26](#_Toc32937216)

[**Sviluppo di PC Ready** 27](#_Toc32937217)

[Riepilogo veloce delle iterazioni 27](#_Toc32937218)

[Modello di dominio 28](#_Toc32937219)

[**SSD e Contratti** 30](#_Toc32937220)

[UC1 30](#_Toc32937221)

[UC2 32](#_Toc32937222)

[UC3 34](#_Toc32937223)

[UC4 35](#_Toc32937224)

[UC5 37](#_Toc32937225)

[UC6 40](#_Toc32937226)

[UC7 40](#_Toc32937227)

[UC8 41](#_Toc32937228)

[UC9 42](#_Toc32937229)

[UC10 43](#_Toc32937230)

[**Progettazione** 44](#_Toc32937231)

[Diagramma delle classi di progetto 45](#_Toc32937232)

[Diagrammi di sequenza di sistema 45](#_Toc32937233)

[SSD - UC1 45](#_Toc32937234)

[SSD – UC2 49](#_Toc32937235)

[SSD – UC3 49](#_Toc32937236)

[SSD – UC4 50](#_Toc32937237)

[SSD – UC5 51](#_Toc32937238)

[SSD – UC6 54](#_Toc32937239)

[SSD – UC7 54](#_Toc32937240)

[SSD – UC8 55](#_Toc32937241)

[SSD – UC9 55](#_Toc32937242)

[SSD - UC10 56](#_Toc32937243)

[**Logica della fase di testing** 57](#_Toc32937244)

[File AccessoTest 57](#_Toc32937245)

[File AcquistoHandlerTest 59](#_Toc32937246)

[File CompatibilityCheckerTest 61](#_Toc32937247)

[File ConfigurationHanlderTest 62](#_Toc32937248)

[File GestisciComponentiHandlerTest 63](#_Toc32937249)

[**Persistenza dei dati: logica del Parser** 65](#_Toc32937250)

[Il database d’esempio 66](#_Toc32937251)

[La classe Parser 67](#_Toc32937252)

[Le funzioni di Caricamento 68](#_Toc32937253)

[Le funzioni di Salvataggio 69](#_Toc32937254)

[Le funzioni di utility 69](#_Toc32937255)

[**Interfaccia Utente: logica della Console** 70](#_Toc32937256)

[La classe Comando 71](#_Toc32937257)

[La classe ElencoComandi 72](#_Toc32937258)

[La classe Console 72](#_Toc32937259)

[Le funzioni di logica 73](#_Toc32937260)

[Le funzioni di utility 73](#_Toc32937261)

# **Ideazione e analisi dei requisiti**

## Requisiti

Il proprietario di un negozio di informatica vuole permettere ai propri clienti di assemblare il proprio PC desktop, facendo scegliere loro le componenti desiderate, mediante un’ applicazione di facile utilizzo.

Tale applicazione deve essere in grado di mostrare al cliente tutte le componenti per PC per categoria, mostrando per ciascuna componente il prezzo, la disponibilità, la compatibilità con altre componenti (Esempio: CPU AMD Ryzen 3 compatibile con socket AM4) e il consumo energetico.

Il proprietario del negozio, gradirebbe che l’applicazione impedisca al cliente di assemblare un PC con componenti incompatibili, o con consumo energetico superiore rispetto al PSU scelto.

Oltre alle singole componenti, l’applicazione deve prevedere delle soluzioni già pronte, ideate dal proprietario per i clienti che non sanno con quali componenti dover assemblare un desktop: tali soluzioni devono essere divise in base alla fascia di prezzo – prestazioni offerte.

Si vuole consentire ai clienti di visionare dei preventivi sui pezzi/sulla configurazione scelta senza obbligo d’acquisto: saranno richiesti i dati del cliente (dati anagrafici, indirizzo e-mail e indirizzo di spedizione) solo se l’acquisto viene confermato.

Il proprietario, grazie ai suoi accordi con i dovuti istituti bancari, vuole consentire ai propri clienti di pagare con la loro carta di credito, con carte prepagate, ed eventualmente mediante contrassegno.

Infine, i prezzi dei singoli componenti sono basati su un listino esistente, tuttavia, se un componente non è presente in magazzino potrebbe essere aggiunto un sovrapprezzo dovuto alla necessità di ordinare il pezzo: tale informazione deve essere notificata all’acquirente.

## Caso d’uso 1: Crea configurazione cliente

|  |  |
| --- | --- |
| **Nome del caso d’uso** | **UC1: Crea configurazione cliente** |
| **Portata** | Applicazione PC Ready |
| **Livello** | Obiettivo utente |
| **Attore primario** | Cliente |
| **Parti interessate e interessi** | * **Cliente:** vuole creare una configurazione PC desktop personalizzata scegliendo i pezzi da lui desiderati * **Sistema:** vuole mostrare al cliente tutte le componenti disponibili o ordinabili per comporre un PC desktop. |
| **Pre-condizioni** | Il cliente ha selezionato l’opzione di creazione di una configurazione. |
| **Garanzia di successo** | La creazione della configurazione si conclude con successo se:   * Non vi sono incompatibilità tra le componenti della configurazione * Il PSU (alimentatore) scelto rientra nei consumi energetici della configurazione * Sono presenti tutti i pezzi fondamentali (CPU – GPU – Motherboard – Storage – Memoria – Case) per il corretto funzionamento del PC desktop |
| **Scenario principale di successo** | 1. Il cliente seleziona l’opzione per la creazione di un PC desktop. 2. Il sistema inizializza una nuova configurazione vuota, impostando dei valori di default per prezzo, nome e consumo energetico. 3. Il sistema restituisce al cliente tutte le categorie di componenti 4. Il cliente seleziona una categoria di componenti 5. Il sistema restituisce al cliente una lista di tutte le componenti della categoria selezionata 6. Il cliente seleziona un prodotto della lista di componenti 7. Il sistema restituisce al cliente tutte le informazioni relative al componente selezionato: per ciascun componente verranno mostrati il prezzo, il consumo energetico, le eventuali specifiche di compatibilità a seconda della categoria del componente stesso, e una breve descrizione. 8. Il cliente aggiunge il componente scelto alla configurazione 9. Il sistema esegue il controllo della compatibilità del componente scelto con i componenti già scelti per la configurazione, e in caso di successo aggiorna i valori di prezzo e di consumo energetico della configurazione e li mostra al cliente   I passi da 4 a 9 vengono ripetuti finché servono   1. Il cliente seleziona l’opzione per confermare il termine “dell’assemblaggio” della configurazione. 2. Il sistema esegue i controlli sulla presenza dei componenti fondamentali, e controlli sul consumo energetico ai fini di garantire al cliente che la configurazione da lui creata possa essere effettivamente funzionante. Conclusi tali controlli il sistema mostra al cliente il costo totale della configurazione e un riepilogo delle componenti selezionate. 3. Il cliente conferma la configurazione da lui creata. |
| **Estensioni** | **a9:** Il cliente vuole rimuovere un componente non più gradito   1. Il cliente seleziona la componente da rimuovere dalla configurazione attuale 2. Il sistema conferma la rimozione, aggiornando i valori di prezzo e consumo energetico della configurazione, dunque, consente al cliente la scelta di un nuovo componente (ripresa passi 4 e 5 del flusso principale)   **a9:** Il sistema rileva una incompatibilità del componente da inserire   1. Il sistema cerca una componente alternativa compatibile con le componenti già scelte dal cliente. 2. Il cliente può cercare una nuova componente, e decidere se seguire il consiglio fornitogli dal sistema, o scegliere una nuova componente. 3. Il sistema esegue un nuovo controllo di compatibilità.   I punti 1 a 3 si ripetono fino a quando non viene inserita una componente compatibile.   1. Il sistema aggiorna i valori di prezzo e consumo energetico della configurazione attuale, e aggiunge il nuovo componente sostitutivo (ripresa passi 4 e 5 del flusso principale).   **b11:** Il sistema rileva un consumo energetico eccessivo rispetto alla potenza massima fornita dal PSU scelto   1. Il sistema mostra al cliente un PSU compatibile con il consumo energetico della sua configurazione, e rimuove il precedente risultato inadeguato. 2. Il cliente cerca un nuovo PSU, e decide se seguire il consiglio fornitogli dal sistema, o scegliere un PSU diverso. 3. Il sistema sostituisce il vecchio PSU, con quello nuovo scelto dal cliente.   **c11:** Il sistema rileva la mancanza di una delle componenti fondamentali per il funzionamento del PC desktop   1. Il sistema mostra la categoria di componenti mancante al cliente, impedendo la conferma della configurazione. 2. Il sistema consente al cliente la scelta di un nuovo componente sostitutivo (ripresa passi 4 e 5 del flusso principale) |
| **Requisiti speciali** | Nessun requisito speciale individuato. |
| **Tecnologie adoperate** | Nessuna tecnologia specifica individuata. |
| **Problemi aperti** | * Come fa il sistema a riconoscere un’incompatibilità tra componenti della configurazione? |

## Caso d’uso 2: Crea bundle

|  |  |
| --- | --- |
| **Nome del caso d’uso** | **UC2: Crea bundle** |
| **Portata** | Applicazione PC Ready |
| **Livello** | Obiettivo utente |
| **Attore primario** | Amministratore del sistema |
| **Parti interessate e interessi** | * **Amministratore:** vuole creare una configurazione PC desktop, direttamente acquistabile da un cliente, inserendo un prezzo che possa differire dall’acquisto delle singole componenti. * **Sistema:** vuole mostrare all’amministratore tutte le componenti disponibili per assemblare il nuovo PC desktop, e al termine inserire quest’ultimo tra i “bundle acquistabili”. |
| **Pre-condizioni** | L’amministratore ha selezionato l’opzione di creazione di un “bundle acquistabile”. |
| **Garanzia di successo** | La creazione del bundle acquistabile si conclude con successo se:   * Non vi sono incompatibilità tra le componenti della configurazione creata. * Il PSU (alimentatore) scelto rientra nei consumi energetici della configurazione assemblata. * Sono presenti tutti i pezzi fondamentali (CPU – GPU – Motherboard – Storage – Memoria – Case) per il corretto funzionamento del PC desktop |
| **Scenario principale di successo** | 1. Un amministratore seleziona l’opzione per la creazione di un PC desktop. 2. Il sistema inizializza una nuova configurazione vuota, impostando dei valori di default per prezzo e consumo energetico. 3. Il sistema restituisce all’amministratore tutte le categorie di componenti 4. L’amministratore seleziona una categoria di componenti 5. Il sistema restituisce all’amministratore una lista di tutte le componenti della categoria selezionata 6. L’amministratore seleziona un prodotto della lista di componenti 7. Il sistema restituisce all’amministratore tutte le informazioni relative al componente selezionato: per ciascun componente verranno mostrati il prezzo, il consumo energetico, le eventuali specifiche di compatibilità a seconda della categoria del componente stesso, e una breve descrizione. 8. L’amministratore seleziona un componente da aggiungere alla configurazione. 9. Il sistema esegue il controllo della compatibilità del componente scelto con i componenti già scelti per la configurazione, e in caso di successo aggiorna i valori di prezzo e di consumo energetico della configurazione e li mostra all’amministratore.   I passi 4 e 9 vengono ripetuti finché servono   1. L’amministratore seleziona l’opzione per confermare il termine “dell’assemblaggio” della configurazione. 2. Il sistema esegue i controlli sulla presenza dei componenti fondamentali, e controlli sul consumo energetico ai fini di garantire all’amministratore che la configurazione da lui creata possa essere effettivamente funzionante. Conclusi tali controlli il sistema mostra al cliente il costo totale della configurazione e un riepilogo delle componenti selezionate. 3. L’amministratore conferma la configurazione da lui creata. 4. L’amministratore inserisce il nome, il fattore di sconto e la descrizione del bundle che conterrà la nuova configurazione creata. 5. Il sistema crea un nuovo bundle relativo alla configurazione creata. |
| **Estensioni** | **a9:** L’amministratore vuole rimuovere un componente non più gradito   1. L’amministratore seleziona la componente da rimuovere dalla configurazione attuale. 2. Il sistema conferma la rimozione, aggiornando i valori di prezzo e consumo energetico della configurazione, dunque, consente all’amministratore la scelta di un nuovo componente (ripresa passi 4 e 5 del flusso principale).   **a9:** Il sistema rileva una incompatibilità del componente da inserire   1. Il sistema cerca una componente alternativa compatibile con le componenti già scelte dall’amministratore. 2. L’amministratore può cercare una nuova componente, e decidere se seguire il consiglio fornitogli dal sistema, o scegliere una nuova componente. 3. Il sistema esegue un nuovo controllo di compatibilità.   I punti 1 a 3 si ripetono fino a quando non viene inserita una componente compatibile.   1. Il sistema aggiorna i valori di prezzo e consumo energetico della configurazione attuale, e aggiunge il nuovo componente sostitutivo (ripresa passi 4 e 5 del flusso principale).   **b11:** Il sistema rileva un consumo energetico eccessivo rispetto alla potenza massima fornita dal PSU scelto   1. Il sistema mostra all’amministratore un PSU compatibile con il consumo energetico della sua configurazione, e rimuove il precedente risultato inadeguato. 2. L’amministratore cerca un nuovo PSU, e decide se seguire il consiglio fornitogli dal sistema, o scegliere un PSU diverso. 3. Il sistema sostituisce il vecchio PSU, con quello nuovo scelto dall’amministratore.   **c11:** Il sistema rileva la mancanza di una delle componenti fondamentali per il funzionamento del PC desktop   1. Il sistema mostra la categoria di componenti mancante all’amministratore, impedendo la conferma della configurazione. 2. Il sistema consente all’amministratore la scelta di un nuovo componente sostitutivo (ripresa passi 4 e 5 del flusso principale) |
| **Requisiti speciali** | Nessun requisito speciale individuato. |
| **Tecnologie adoperate** | Nessuna tecnologia specifica individuata. |
| **Problemi aperti** | * Come fa il sistema a riconoscere un’incompatibilità tra componenti della configurazione? * Come fa il sistema a gestire i “bundle acquistabili”? |

## Caso d’uso 3: Inserisci nuovo componente

|  |  |
| --- | --- |
| **Nome del caso d’uso** | **UC3: Inserisci nuovo componente** |
| **Portata** | Applicazione PC Ready |
| **Livello** | Obiettivo utente, CRUD |
| **Attore primario** | Amministratore del sistema |
| **Parti interessate e interessi** | * **Amministratore:** vuole un rapido inserimento di un nuovo componente e della quantità disponibile di quest’ultimo. * **Sistema:** vuole assegnare a ciascun componente delle specifiche dettagliate e generare per ciascuno di questi un codice univoco. |
| **Pre-condizioni** | Il sistema deve essere funzionante e l’amministratore deve averne pieno accesso ed essersi autenticato. |
| **Garanzia di successo** | Viene inserito nel sistema un nuovo componente, al quale viene associato un codice identificativo univoco. |
| **Scenario principale di successo** | 1. L’amministratore richiede al sistema l’inserimento di un nuovo componente. 2. Il sistema chiede all’amministratore le informazioni relative al nuovo componente da inserire. 3. L’amministratore fornisce al sistema nome, categoria, consumo energetico, prezzo e descrizione del componente 4. Il sistema inserisce la nuova componente e chiede all’amministratore il numero di copie del componente da inserire. 5. L’amministratore fornisce al sistema il numero di copie del componente da inserire. 6. Il sistema inserisce un numero di copie del componente pari alla quantità specificata. 7. Il sistema mostra all’amministratore i codici univoci di ciascuna copia componente inserita, e stampa   un’etichetta (contenente il codice univoco) per ciascuna di esse.   1. L’amministratore termina l’operazione di inserimento |
| **Estensioni** | Nessuna estensione individuata. |
| **Requisiti speciali** | Nessun requisito speciale individuato. |
| **Tecnologie adoperate** | Nessuna tecnologia specifica individuata. |
| **Problemi aperti** | * Come fa il sistema a stampare le etichette contenenti il codice univoco dei componenti inseriti? |

## Caso d’uso 4: Inserisci copia componente

|  |  |
| --- | --- |
| **Nome del caso d’uso** | **UC4: Inserisci copia componente** |
| **Portata** | Applicazione PC Ready |
| **Livello** | Obiettivo utente, CRUD |
| **Attore primario** | Amministratore del sistema |
| **Parti interessate e interessi** | * **Amministratore:** vuole un rapido inserimento di una o più copie di un componente già esistente * **Sistema:** vuole assegnare al componente identificato dall’amministratore un numero di nuove copie ben definito, e assegnare a ciascuna delle suddette copie un identificativo univoco. |
| **Pre-condizioni** | La componente indicata dall’amministratore deve essere presente nella memoria del sistema, ovvero il sistema deve possedere un’istanza esistente di componente per poter assegnare ad esso delle nuove copie. |
| **Garanzia di successo** | Viene inserito nel sistema una nuova (o più ) copie di un componente, alle quali viene associato un codice identificativo univoco. |
| **Scenario principale di successo** | 1. L’amministratore richiede al sistema l’inserimento di una nuova copia di un componente, fornendone il codice e la categoria. 2. Il sistema individuato il componente esistente, chiede all’amministratore il numero di copie da inserire. 3. L’amministratore fornisce al sistema la quantità di copie da inserire. 4. Il sistema inserisce un numero di copie del componente pari alla quantità specificata. 5. Il sistema mostra all’amministratore i codici univoci di ciascuna copia componente inserita, e stampa un etichetta (contenente il codice univoco) per ciascuna di esse. 6. L’amministratore termina l’operazione di inserimento |
| **Estensioni** | Nessuna estensione individuata. |
| **Requisiti speciali** | Nessun requisito speciale individuato. |
| **Tecnologie adoperate** | Nessuna tecnologia specifica individuata. |

## Caso d’uso 5: Effettua acquisto

|  |  |
| --- | --- |
| **Nome del caso d’uso** | **UC5: Effettua acquisto** |
| **Portata** | Applicazione PC Ready |
| **Livello** | Obiettivo utente |
| **Attore primario** | Cliente |
| **Parti interessate e interessi** | * **Cliente:** vuole effettuare un acquisto di una componente generica esistente nel sistema. * **Sistema:** vuole mostrare al cliente tutte le componenti disponibili o ordinabili. |
| **Pre-condizioni** | Il cliente ha selezionato l’opzione di acquisto di una componente generica |
| **Garanzia di successo** | Il cliente ha effettuato il pagamento delle componenti scelte e ha finalizzato l’acquisto fornendo tutte le informazioni necessarie per la spedizione dei prodotti. |
| **Scenario principale di successo** | 1. Il cliente seleziona l’opzione di acquisto 2. Il sistema restituisce al cliente un catalogo di tipologie di prodotti disponibili (ovvero categorie di componenti o bundle) 3. Il cliente seleziona una tipologia disponibile 4. Il sistema restituisce al cliente una lista di prodotti in base alla tipologia selezionata dal cliente. 5. Il cliente seleziona un prodotto dalla lista. 6. Il sistema restituisce al cliente tutte le informazioni relative al prodotto selezionato. 7. Il cliente chiede al sistema di aggiungere il prodotto al carrello. 8. Il sistema, dopo aver controllato la presenza di una copia del componente, conferma l’avvenuto inserimento del prodotto nel carrello, aggiorna il costo totale dei prodotti contenuti in quest’ultimo e restituisce al cliente la lista dei prodotti contenuti e il costo totale.   I passi da 1 a 8 vengono ripetuti fino a quando il cliente vuole continuare ad acquistare componenti.   1. Il cliente seleziona l’opzione per l’acquisto degli articoli contenuti nel carrello 2. Il sistema chiede al cliente le informazioni sulla spedizione (indirizzo di spedizione, città, CAP) 3. Il cliente inserisce le informazioni richieste dal sistema 4. Il sistema chiede al cliente la modalità di pagamento, mostrando una lista di possibili opzioni, e chiedendo al cliente le informazioni relative al sistema di pagamento scelto (numero carta, codice a tre cifre) 5. Il cliente fornisce i dati relativi alla modalità di pagamento e conferma l’acquisto 6. Il sistema notifica il successo dell’operazione |
| **Estensioni** | **a8:** Il sistema rileva che non sono presenti copie del componente selezionato dal cliente   1. Il sistema notifica al cliente l’assenza di copie del componente scelto, e impedisce l’inserimento nel carrello del componente 2. Il sistema consente al cliente di scegliere una nuova categoria di prodotti (ripresa dal passo 2 del flusso principale).   **b9:** Il cliente vuole rimuovere un componente non più gradito   1. Il cliente seleziona la componente da rimuovere dal suo carrello attuale 2. Il sistema conferma la rimozione, aggiornando la lista delle componenti presente nel carrello, il numero dei pezzi e il prezzo totale delle componenti, e infine consente al cliente la scelta di un nuovo componente presente in catalogo (ripresa dal passo 2 del flusso principale). |
| **Requisiti speciali** | Nessun requisito speciale individuato. |
| **Tecnologie adoperate** | Nessuna tecnologia specifica individuata. |

## Caso d’uso 6: Registrazione cliente

|  |  |
| --- | --- |
| **Nome del caso d’uso** | **UC6: Registrazione cliente** |
| **Portata** | Applicazione PC Ready |
| **Livello** | Obiettivo utente, CRUD |
| **Attore primario** | Cliente |
| **Parti interessate e interessi** | * **Cliente:** vuole effettuare la registrazione al servizio fornito dall’applicazione PC Ready. * **Sistema:** vuole tutte le informazioni relative al nuovo cliente da registrare |
| **Pre-condizioni** | Non deve esistere l’istanza del suddetto cliente in memoria del sistema PC Ready |
| **Garanzia di successo** | La nuova istanza di cliente viene mantenuta nella memoria del sistema. |
| **Scenario principale di successo** | 1. Il cliente richiede al sistema di poter effettuare una nuova registrazione. 2. Il sistema richiede i dati del cliente (nome, cognome, e-mail, password ecc.). 3. Il cliente inserisce tutti i suoi dati. 4. Il sistema controlla se la mail inserita dall’utente è già presente nel sistema, e se la password coincide con il campo “conferma password” 5. Il sistema notifica al cliente l’avvenuta registrazione mostrando un riepilogo delle informazioni non sensibili |
| **Estensioni** | **a4:** Il sistema rileva una mail già esistente in memoria   1. Il sistema notifica al cliente che la mail è già in uso e chiede al cliente nuovi dati per la registrazione 2. Il cliente inserisce un nuovo indirizzo e-mail per completare la registrazione 3. Il sistema controlla nuovamente l’indirizzo inserito   **a4:** Il sistema rileva che il campo password e il campo conferma password non coincidono   1. Il sistema notifica al cliente che i campi password non coincidono e chiede al cliente nuovi dati per la registrazione 2. Il cliente inserisce nuovi dati per la registrazione 3. Il sistema controlla nuovamente i dati inseriti dall’utente |
| **Requisiti speciali** | Nessun requisito speciale individuato. |
| **Tecnologie adoperate** | Nessuna tecnologia specifica individuata. |

## Caso d’uso 7: Login

|  |  |
| --- | --- |
| **Nome del caso d’uso** | **UC7: Login** |
| **Portata** | Applicazione PC Ready |
| **Livello** | Obiettivo utente, CRUD |
| **Attore primario** | Utente generico |
| **Parti interessate e interessi** | * **Utente generico:** vuole effettuare il login per usufruire del servizio fornito dall’applicazione PC Ready. * **Sistema:** vuole tutte le informazioni relative all’utente da autenticare |
| **Pre-condizioni** | Deve esistere l’istanza del suddetto utente in memoria del sistema PC Ready |
| **Garanzia di successo** | L’utente generico viene riconosciuto come un cliente o come amministratore, e può usufruire del servizio |
| **Scenario principale di successo** | 1. L’utente generico richiede al sistema di poter effettuare un login, specificando se vuole eseguire il login come amministratore o come cliente 2. Il sistema richiede i dati dell’utente (e-mail e password). 3. Il cliente inserisce la tipologia di login e tutti i suoi dati. 4. Il sistema notifica al cliente l’avvenuto login mostrando un messaggio di benvenuto |
| **Estensioni** | **a4:** Il sistema rileva che i dati inseriti dall’utente sono errati   1. Il sistema mostra all’utente un messaggio di errore di login e invita l’utente ad inserire nuovamente le sue credenziali |
| **Requisiti speciali** | Nessun requisito speciale individuato. |
| **Tecnologie adoperate** | Nessuna tecnologia specifica individuata. |

## Caso d’uso 8: Rimozione componente

|  |  |
| --- | --- |
| **Nome del caso d’uso** | **UC8: Rimozione componente** |
| **Portata** | Applicazione PC Ready |
| **Livello** | Obiettivo utente, CRUD |
| **Attore primario** | Amministratore del sistema |
| **Parte interessate e interessi** | * **Amministratore:** vuole eliminare un componente presente nel catalogo in quanto non più utile o disponibile. * **Sistema:** vuole tutte le informazioni relative al componente che l’amministratore vuole rimuovere dal catalogo |
| **Pre-condizioni** | L’amministratore deve avere effettuato il login al servizio, e la sua istanza deve essere stata recuperata. |
| **Garanzia di successo** | La componente individuata mediante i dati forniti è stata eliminata. |
| **Scenario principale di successo** | 1. L’amministratore richiede al sistema di poter rimuovere un componente dai prodotti disponibili. 2. Il sistema richiede il codice del componente da rimuovere. 3. L’amministratore fornisce al sistema il codice del componente da rimuovere. 4. Il sistema verifica la presenza del componente e lo rimuove insieme alle sue copie eventualmente presenti. 5. Il sistema rimuove Bundle e Configurazioni che contengono il componente appena eliminato |
| **Estensioni** | **a4:** Il sistema rileva che il codice fornito dall’amministratore non corrisponde a nessun componente esistente   1. Il sistema mostra all’amministratore un messaggio di errore e chiede l’inserimento di un nuovo codice componente valido 2. L’amministratore fornisce al sistema un nuovo codice del componente da rimuovere (ripresa del punto 3 del flusso principale) |
| **Requisiti speciali** | Nessun requisito speciale individuato. |
| **Tecnologie adoperate** | Nessuna tecnologia specifica individuata. |

## Caso d’uso 9: Crea promozione

|  |  |
| --- | --- |
| **Nome del caso d’uso** | **UC9: Crea promozione** |
| **Portata** | Applicazione PC Ready |
| **Livello** | Obiettivo utente, CRUD |
| **Attore primario** | Amministratore del sistema |
| **Parte interessate e interessi** | * **Amministratore:** vuole inserire una promozione per un prodotto, ovvero vuole metterlo in sconto. * **Sistema:** vuole tutte le informazioni relative al componente che l’amministratore vuole mettere in promozione |
| **Pre-condizioni** | Deve esistere l’istanza del componente da mettere in promozione in memoria del sistema PC Ready |
| **Garanzia di successo** | Viene modificato il fattore di promozione del componente selezionato dall’amministratore |
| **Scenario principale di successo** | 1. L’amministratore richiede al sistema di inserire una nuova promozione. 2. Il sistema richiede all’amministratore il prodotto sul quale applicare la promozione. 3. L’amministratore inserisce il codice del prodotto in promozione. 4. Il sistema chiede all’amministratore la percentuale di sconto da applicare. 5. L’amministratore inserisce la percentuale di sconto. 6. Il sistema applica percentuale di sconto ai prodotti indicatogli e comunica all’admin il costo dei prodotti scontati. |
| **Estensioni** | **a4:** Il sistema rileva che il codice fornito dall’amministratore non corrisponde a nessun componente esistente   1. Il sistema mostra all’amministratore un messaggio di errore e chiede l’inserimento di un nuovo codice componente valido 2. L’amministratore fornisce al sistema un nuovo codice del componente da rimuovere (ripresa del punto 3 del flusso principale) |
| **Requisiti speciali** | Nessun requisito speciale individuato. |
| **Tecnologie adoperate** | Nessuna tecnologia specifica individuata. |

## Caso d’uso 10: Modifica ordine

|  |  |
| --- | --- |
| **Nome del caso d’uso** | **UC10: Modifica ordine** |
| **Portata** | Applicazione PC Ready |
| **Livello** | Obiettivo utente, CRUD |
| **Attore primario** | Cliente |
| **Parte interessate e interessi** | * **Cliente:** vuole modificare le informazioni dell’indirizzo di spedizione associato ad un ordine effettuato in precedenza * **Sistema:** vuole conoscere tutte le informazioni relative all’ordine da modificare |
| **Pre-condizioni** | Il cliente deve essersi autenticato, e deve esistere un’istanza di ordine associata ad esso. |
| **Garanzia di successo** | Le informazioni relative all’ordine individuato sono state modificate. |
| **Scenario principale di successo** | 1. Il cliente chiede al sistema i suoi ordini effettuati 2. Il sistema restituisce al cliente i suoi ordini, e chiede al cliente l’identificativo dell’ordine da modificare e quali informazioni dell’ordine vuole modificare (indirizzo di spedizione, città, ecc..). 3. Il cliente fornisce al sistema le nuove informazioni sulla spedizione e l’identificativo dell’ordine da modificare 4. Il sistema registra le modifiche e conferma il buon esito dell’operazione. |
| **Estensioni** | **a2:** Il sistema rileva che il cliente non ha effettuato nessun ordine   1. Il sistema mostra all’utente un messaggio di errore e termina l’operazione.   **b4:** Il sistema rileva che il cliente ha inserito un codice ordine non valido   1. Il sistema mostra all’utente un messaggio di errore e chiede nuovamente all’utente l’inserimento di un nuovo identificativo dell’ordine da modificare e quali informazioni dell’ordine vuole modificare 2. Il cliente fornisce al sistema le nuove informazioni sulla spedizione e l’identificativo dell’ordine da modificare (ripresa del punto 3 del flusso principale) |
| **Requisiti speciali** | Nessun requisito speciale individuato. |
| **Tecnologie adoperate** | Nessuna tecnologia specifica individuata. |

## Regole di dominio

1. Una configurazione PC desktop necessita delle seguenti componenti fondamentali per un corretto funzionamento:
   * CPU
   * GPU
   * Motherboard
   * Storage
   * Memoria
   * Case
2. Una Motherboard si dice incompatibilecon:
   * CPU: se il socket della motherboard non è uguale a quello della CPU.
   * RAM: se la tipologia della RAM non è supportata dalla motherboard.
   * Case: se il form factor del case non è adatto a contenere la motherboard.
3. Una GPU si dice incompatibile con:
   * Case: se il case non ha abbastanza slot laterali per contenere la GPU.
4. Un PSU si dica incompatibile con:
   * Case: se lo slot per l’alimentazione del case è inadatto al form factor del PSU.
5. Una configurazione PC desktop necessita di un Power Supply (alimentatore) la cui potenza fornita deve essere uguale o inferiore alla somma dei consumi energetici di tutte le componenti.

## Glossario

|  |  |
| --- | --- |
| **CPU** | **Central Processing Unit**, è il “cuore pulsante” di un qualsiasi PC Desktop in quanto coordina in maniera centralizzata tutte le altre unità di elaborazione presenti, si occupa della gestione delle varie periferiche interne o schede elettroniche come schede audio, schede di rete o schede video |
| **GPU** | **Graphic Processing Unit**, è l'unità di elaborazione grafica conosciuta anche come unità di elaborazione visiva (VPU, in inglese visual processing unit) o processore grafico o processore visivo.  È una tipologia particolare di coprocessore che si contraddistingue per essere specializzato nel rendering di immagini grafiche. |
| **APU** | **Accelerated processing unit** (unità d'elaborazione accelerata) o APU, nel suo uso più comune si riferisce ad un singolo die che combina componenti CPU, GPU e PCIe: difatti in gergo comune si indica con APU un “processore con scheda grafica integrata”.  La GPU integrata in tale APU è solitamente meno performante rispetto ad una generica GPU integrata. |
| **Motherboard** | Una **scheda madre** detta anche scheda di sistema, in lingua inglese motherboard ("scheda madre") mainboard ("scheda principale") o in gergo “mobo”, è un tipo di scheda elettronica principale che raccoglie in sé tutta la circuiteria elettronica e i collegamenti di interfaccia tra i vari componenti interni principali di un personal computer come memoria e le altre schede elettroniche montate o alloggiate sopra, comprendendo anche i bus di espansione e le interfacce verso le periferiche esterne.  Le motherboard “consumer” per PC desktop sono disponibili in diversi formati, tra cui i più comuni sono:   * **ATX**: dimensione (305×244) * **micro ATX**: dimensione (244×244) * **mini ATX**: dimensione (150×150) |
| **PSU** | Un alimentatore, meglio conosciuto con il suo nome inglese **power supply unit (PSU)**, è un dispositivo elettrico che fornisce energia elettrica a un carico elettrico.  La funzione principale di un alimentatore è quella di convertire l'alimentazione CA in corrente continua regolata a bassa tensione per i componenti interni di un computer.  I PSU sono divisi per categorie di efficienza (dal meno al più efficiente) secondo il programma di efficienza 80 Plus:   * 80 Plus “white” * 80 Plus Bronze * 80 Plus Silver * 80 Plus Gold * 80 Plus Platinum |
| **Efficienza di un PSU** | Per spiegare cosa vuol dire efficienza per un PSU è necessario introdurre un breve esempio: si ipotizzi che un PSU eroghi 300W al sistema, ma in realtà assorba 375W dalla presa a muro.  Questo significa che l'efficienza è 375W/300W, che equivale allo 0,8 o all'80%: quei 75W non producono altro se non **calore**.  Più efficiente è un alimentatore, meno calore produce internamente e certamente minore è il suo consumo energetico. La quantità di calore prodotta è cruciale in quanto impatta sulle prestazioni dell'alimentatore e tutti i suoi componenti interni. Inoltre, oltre a questo, maggiore è il carico termico, più difficile è il compito del sistema di raffreddamento dell'alimentatore. |
| **Memoria RAM** | In elettronica e informatica la RAM (**Random Access Memory** ovvero memoria ad accesso casuale) è un tipo di memoria volatile caratterizzata dal permettere l'accesso diretto a qualunque indirizzo di memoria con lo stesso tempo di accesso.  Nella memoria RAM vengono copiati (caricati) i programmi che la CPU deve eseguire. |
| **Storage SSD** | Una unità di **memoria a stato solido** (in acronimo SSD dal corrispondente termine inglese solid-state drive), in elettronica e informatica, è un dispositivo di memoria di massa basato su semiconduttori, che utilizza memoria allo stato solido, in particolare memoria flash, per l'archiviazione dei dati.  A differenza di una memoria, un SSD in quanto esente da parti meccaniche comporta una serie di vantaggi:   * Rumorosità assente * Minore possibilità di guasto * Non necessitano di deframmentazione * Maggiore velocità di trasferimento dati * Minore produzione di calore |
| **Storage HDD** | Un disco rigido, detto anche **disco fisso** (abbreviato comunemente in hard disk con la sigla HDD), in elettronica e informatica indica un dispositivo di memoria di massa di tipo magnetico che utilizza uno o più dischi magnetizzati per l'archiviazione di dati e applicazioni (file, programmi e sistemi operativi). |
| **Configurazione** | Nel contesto in questione, per **configurazione** si intende un insieme di componenti tale da poter costituire un PC pienamente funzionante. |
| **Socket CPU** | Nell'hardware del computer, un **socket CPU** o uno slot CPU, contiene uno o più componenti meccanici che forniscono connessioni meccaniche ed elettriche tra un microprocessore (CPU) e la scheda madre stessa.  Ciò consente di posizionare e sostituire l'unità di elaborazione centrale (CPU) senza saldatura. |
| **Bundle** | PC desktop già assemblato, formato da una combinazione di più componenti, messo in vendita ad un prezzo vantaggioso. |
| **Form factor** | Termine utilizzato per indicare delle dimensioni standardizzate di una motherboard, di un PSU o di un Case. |

# **Documento di visione**

## Introduzione

L’obiettivo di questo elaborato è la creazione di un applicativo di nome “PC Ready”, che consenta agli acquirenti di un negozio di informatica di acquistare componenti per PC desktop, PC desktop pre-assemblati o di creare ed acquistare un PC con le componenti da questi desiderate.

## Scopo

Lo scopo del documento di visione è quello di definire, sottolineare e analizzare le peculiarità, le necessità e i servizi offerti da “PC Ready”.

In tale documento si possono osservare tutte le parti interessate e le loro necessità, ricordando che maggiori dettagli riguardo al soddisfacimento di queste ultime sono contenuti nel Modello dei casi d’uso.

## Portata

Il documento di visione ivi presente è relativo all’elaborato “PC Ready”, stilato e revisionato dagli studenti B.Caruso, G.Fallica e G.Costanzo.

L’obiettivo principale del gruppo è quello di soddisfare tutte le richieste del committente, così come tutti i requisiti non funzionali individuati nel corso dell’elaborato.

## Definizioni, acronimi e abbreviazioni

Per tutte le informazioni relative a definizioni, sigle, abbreviazioni e altro ancora, si faccia riferimento al Glossario posto sopra al documento di visione.

## Posizionamento

**Opportunità di business**: l’applicativo che verrà sviluppato consentirà al proprietario del negozio di informatica di raggiungere più clienti, fornendo loro un servizio di e-shop con la possibilità di ricevere comodamente a casa i prodotti acquistati.

Inoltre, tale applicativo permetterà al proprietario del negozio in questione di applicare promozioni ai prodotti inseriti mediante una semplice gestione di tutti i componenti presenti in magazzino.

La tabella posta in seguito mostra la **formulazione del problema:**

|  |  |
| --- | --- |
| **Descrizione del problema** | Il sistema precedentemente usato dal proprietario del negozio di informatica non permette a questi di poter acquisire una maggiore clientela.  Quando un cliente commissiona un PC Desktop, in quanto non vi è un meccanismo di controllo automatico sulla compatibilità delle componenti, si ha una probabilità elevata di comporre una configurazione non funzionante. |
| **Attori coinvolti** | Amministratore del sistema PC Ready, Clienti generici |
| **Impatto** | Processo di creazione di un PC desktop soggetto ad errori di compatibilità o insufficienza energetica, con conseguente danneggiamento delle componenti nei casi più gravi e insoddisfazione dei clienti. |
| **Benefici in caso di successo** | Informatizzazione della creazione di un PC desktop, con conseguente riduzione del tasso degli errori, e aumento della clientela raggiungibile garantito dal processo di vendita online. |

La tabella seguente mostra la **formulazione della posizione del prodotto:**

|  |  |
| --- | --- |
| **Destinatari** | Il prodotto è rivolto ad un negozio di informatica specializzato nella vendita di componenti per PC desktop |
| **Obbiettivi** | Il committente ha la necessità di un applicativo che consenta di effettuare delle vendite online, e che controlli la compatibilità delle componenti nel caso in cui si voglia assemblare un PC Desktop, azzerando la probabilità di commettere errori in tale fase. |
| **Tipologia** | Prodotto software personalizzato per la vendita, ma generale riguardo i controlli di compatibilità delle componenti PC. |
| **Funzione** | “PC Ready” offre una soluzione di gestione delle vendite online, integrata ad un servizio di controllo compatibilità delle componenti. |
| **Soluzioni alternative attuali** | Una soluzione simile per il controllo della compatibilità delle componenti risulta essere “PC Part Picker”, tuttavia questo applicativo web non consente di gestire le vendite o il magazzino di un negozio di informatica. |
| **Caratteristiche del prodotto** | Il prodotto in questione offre un livello di personalizzazione adatto ai prodotti venduti dal committente, aggiungendo delle funzionalità di controllo che agevolano il lavoro del committente e garantiscono ai clienti PC Desktop pronti all’uso. |

## Parti interessate e descrizioni utente

Ecco una breve lista delle parti interessate:

* **Committente:** è colui che ha commissionato lo sviluppo del prodotto.

Il suo compito è quello di fornire agli sviluppatori quante più informazioni possibili (requisiti) sul prodotto richiesto e feedback riguardo ad ogni versione dell’applicativo.

* **Amministratore del sistema:** è colui che si occupa della gestione del magazzino, della gestione delle promozioni e dei bundle acquistabili dai clienti.

Il suo compito è quello di utilizzare in prima persona l’applicativo sviluppato, fornendo feedback dettagliati agli sviluppatori.

* **Sviluppatore software:** è il responsabile dello sviluppo dell’applicativo.

Il suo compito è quello di progettare il prodotto software, cercando di soddisfare tutte le richieste del committente.

# **Sviluppo di PC Ready**

## Riepilogo veloce delle iterazioni

Lo sviluppo di PC Ready è stato svolto in modo iterativo, in modo da apportare delle modifiche graduali e ben controllate, ridurre gli errori di sviluppo, ed eventualmente permettere una più facile correzione di quest’ultimi.

La realizzazione dell’applicazione è stata articolata in quattro iterazione che hanno permesso di coprire la totalità dei casi d’uso mostrati nella parte iniziale di questo documento.

Ecco un breve dettaglio dei contenuti delle iterazioni:

* Iterazione 1
  + Sviluppo dei casi d’uso fondamentali che consentono ad amministratore ed utente di creare una configurazione (senza effettuare alcun controllo di compatibilità sulle componenti inserite), ovvero di UC1 e UC2
  + Sviluppo dei casi d’uso che consentono ad un amministratore di inserire nuove componenti e copie di quest’ultime, ovvero di UC3 e UC4
  + Sviluppo dei diagrammi di sequenza di sistema e dei diagrammi di sequenza per UC1,UC2,UC3,UC4
  + Implementazione dei casi d’uso sopra riportati
* Iterazione 2
  + Sviluppo della logica di compatibilità delle componenti di una configurazione così come previsto dalle estensioni di UC1 e UC2.
  + Riduzione delle responsabilità della classe SistemaPCReady mediante l’uso di controller di caso d’uso (by pattern Controller).
  + Rimozione della classe Categoria e introduzione della classe Catalogo, con conseguente introduzione dell’attributo “categoria” per la classe Componente.
  + Applicazione del pattern Composite per la gestione della Configurazione e delle specializzazioni della classe Componente.
  + Correzioni e migliorie del codice dell’iterazione precedente.
  + Implementazione della console comandi
    - Per maggiori informazioni riguardo a tale iterazioni si consiglia di consultare il documento “Discussione Iterazione 2”.
* Iterazione 3
* Sviluppo dei casi d’uso UC5, UC6 e UC7 in forma completa
* Aggiornamento del modello di dominio e diagramma delle classi di progetto con le nuove classi introdotte dai casi d’uso sopra citati
* Sviluppo dei diagrammi di sequenza di sistema e dei diagrammi di sequenza per UC5,UC6,UC7
* Implementazione dei casi d’uso UC5, UC6, UC7 con le rispettive estensioni
* Migliorie dell’interfaccia utente e aggiunta dei nuovi comandi di login, registrazione e acquisto
  + - Per maggiori informazioni riguardo a tale iterazioni si consiglia di consultare il documento “Discussione Iterazione 3”.
* Iterazione 4
  + Sviluppo dei casi d’uso UC8, UC9 e UC10 in forma completa
  + Sviluppo dei diagrammi di sequenza di sistema e dei diagrammi di sequenza per UC8,UC9,UC10
  + Implementazione di UC8,UC9,UC10 con le relative estensioni
  + Migliorie all’interfaccia utente, con aggiunta di nuovi comandi
  + Raffinamento dei test case con il comando: “@FixMethodOrder(MethodSorters.***NAME\_ASCENDING***)”

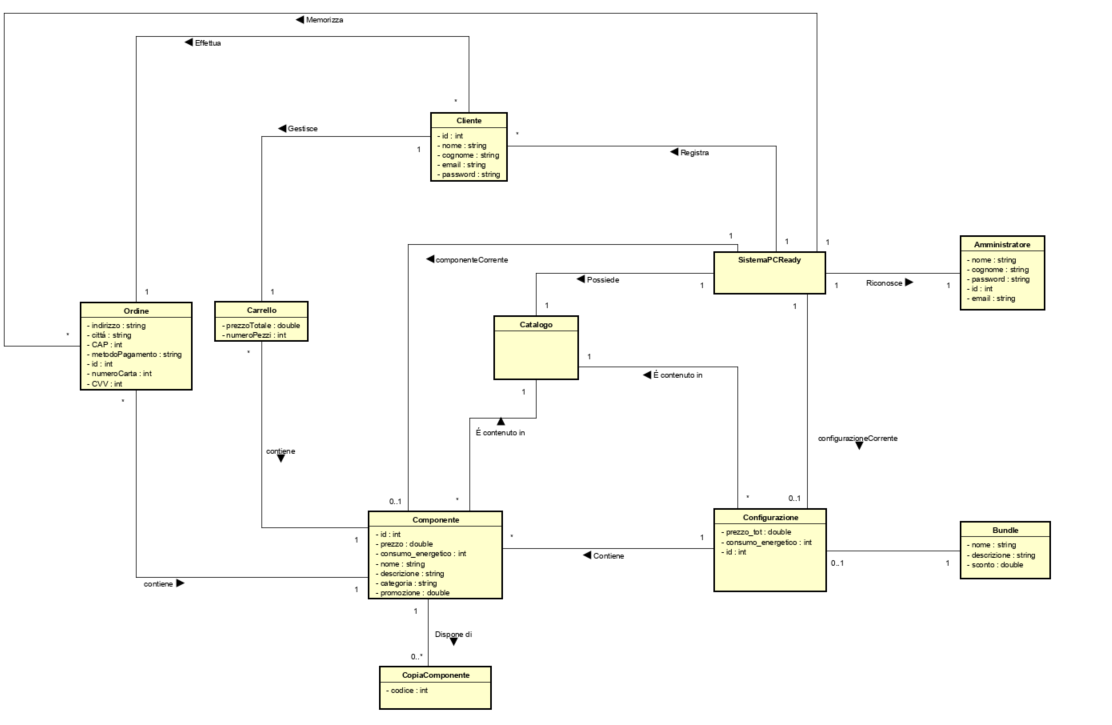
## Modello di dominio

Il modello di dominio è un diagramma che consente di visualizzare tutte le classi concettuali individuate durante l’analisi del problema e lo sviluppo del progetto.

Mediante il suddetto diagramma, che risulta essere un elaborato fondamentale per una buona analisi orientata agli oggetti, mette in evidenzia tutti gli attributi delle classi concettuali, e le relazioni di maggior rilievo che si instaurano tra quest’ultime.

Ovviamente, anche il modello di dominio è stato sviluppato in modo iterativo, per cui di volta in volta sono state applicate ad esso tutte le novitá o le modifiche previste dall’iterazione corrente.

A seguire viene mostrato il modello di domino “finale”, ovvero il modello di dominio che si è ottenuto dopo l’ultima iterazione sviluppata:



Per apprezzare meglio il suddetto diagramma, si consiglia di consultare il file Astah in allegato alla documentazione.

Dunque, mediante tale diagramma sono state individuate le seguenti classi concettuali:

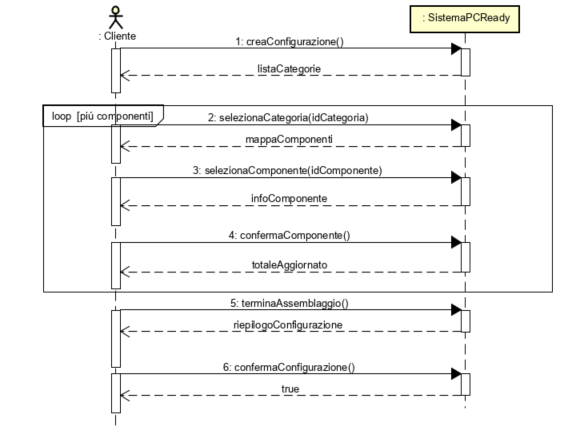
* **SistemaPCReady:** rappresenta il sistema PC Ready;
* **Cliente:** rappresenta un cliente generico, che intende creare configurazioni personalizzate ed acquistarle, o acquistare delle componenti singole o bundle;
* **Amministratore:** rappresenta un amministratore del sistema;
* **Catalogo:** rappresenta il catalogo di componenti disponibili, configurazioni create dai clienti, e bundle pronti per l’acquisto;
* **Carrello:** rappresenta una lista di articoli che il cliente ha selezionato per l’acquisto, difatti il nome di tale classe è un’analogia con un carrello reale;
* **Componente:** rappresenta un componente informatico, come una RAM, una GPU e altro ancora;
* **CopiaComponente:** rappresenta una copia di un componente esistente, attualmente contenuta nel magazzino del negozio;
* **Ordine:** rappresenta un ordine effettuato da un cliente, e contiene tutte le informazioni necessarie per la spedizione dei prodotti acquistati dal cliente;
* **Configurazione:** rappresenta una configurazione assemblata virtualmente da un cliente;
* **Bundle:** rappresenta una configurazione realmente assemblata da un amministratore, alla quale viene assegnato un ulteriore fattore di promozione.

# **SSD e Contratti**

In questo capitolo verranno mostrati tutti i diagrammi di sequenza di sistema SSD e i contratti delle operazioni,(i quali descrivono con un maggiore grado di dettaglio le operazioni presenti nei diagrammi SSD), che sono stati ottenuti durante l’analisi orientata agli oggetti.

Per facilitare la lettura di tali SSD e contratti, si suddivide tale capitolo in casi d’uso.

## UC1



|  |  |
| --- | --- |
| **Nome del contratto** | **CO1: Crea Configurazione** |
| **Operazione** | creaConfigurazione() |
| **Riferimenti** | Caso d’uso UC1: Crea configurazione cliente |
| **Pre-condizioni** | * Il cliente è stato autenticato |
| **Post-condizioni** | * È stata creata un’istanza Conf di Configurazione * Gli attributi di Conf (prezzo, consumo energetico) sono stati inizializzati ai valori di default) * È stata inizializzata una lista di “Componente” denominata “listaComponenti” (attributo che indica i componenti presenti nella configurazione) |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO2: Seleziona Categoria** |
| **Operazione** | selezionaCategoria() |
| **Riferimenti** | Caso d’uso UC1: Crea configurazione cliente |
| **Pre-condizioni** | * È stata inizializzata un’istanza Conf di Configurazione, e una lista “listaComponenti” di Componente * Il cliente ha visualizzato tutte le categorie di Componente |
| **Post-condizioni** | * Viene inizializzata una mappa idComponente-Componente denominata “mappaCorrente” * Il contenuto di mappaCorrente è stato aggiornato in base alla Categoria scelta |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO3: Seleziona Componente** |
| **Operazione** | selezionaComponente() |
| **Riferimenti** | Caso d’uso UC1: Crea configurazione cliente |
| **Pre-condizioni** | * È stata inizializzata una lista di Componente “listaCorrente” |
| **Post-condizioni** | * È stata inizializzata un’istanza componenteCorrente di Componente |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO4: Conferma Componente** |
| **Operazione** | confermaComponente() |
| **Riferimenti** | Caso d’uso UC1: Crea configurazione cliente |
| **Pre-condizioni** | * È stata inizializzata un’istanza Comp di Componente. |
| **Post-condizioni** | * L’istanza Comp viene associata a Conf mediante l’associazione “Contiene” * Gli attributi prezzo e consumo energetico di Conf vengono aggiornati in base alle specifiche di Comp |

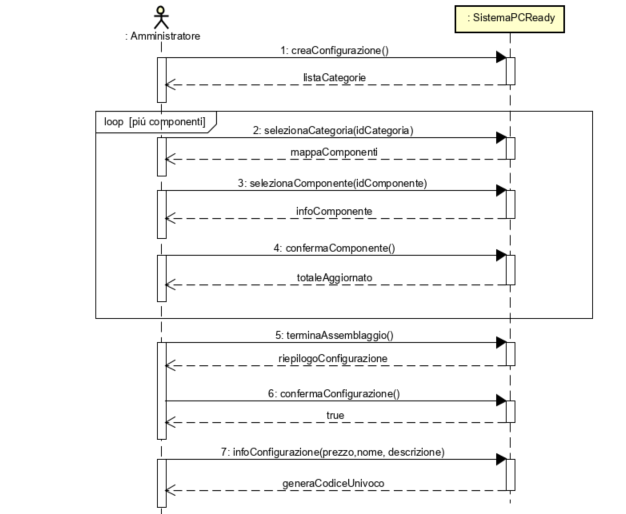
|  |  |
| --- | --- |
| **Nome del contratto** | **CO5: Termina Assemblaggio** |
| **Operazione** | confermaAssemblaggio() |
| **Riferimenti** | Caso d’uso UC1: Crea configurazione cliente |
| **Pre-condizioni** | * Il cliente ha selezionato tutti i componenti per la sua configurazione |
| **Post-condizioni** |  |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO6: Conferma Configurazione** |
| **Operazione** | confermaConfigurazione() |
| **Riferimenti** | Caso d’uso UC1: Crea configurazione cliente |
| **Pre-condizioni** | * Il cliente ha verificato il riepilogo di tutte le scelte che ha effettuato e le specifiche (prezzo e consumo) della configurazione creata |
| **Post-condizioni** | * Il Sistema mantiene in memoria la Configurazione appena creata |

## UC2

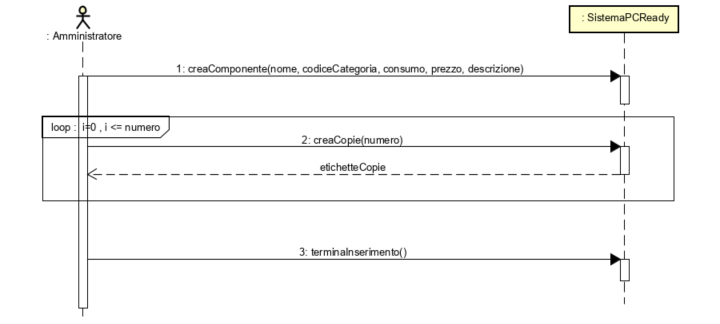
Prima di mostrare SSD e contratti, è d’obbligo sottolineare che molte delle operazioni presenti in tale caso d’uso sono uguali a quello precedente, con l’unica differenza che si sta trattando di un Bundle: dunque, verranno mostrati solo i contratti relativi alle “funzioni aggiuntive”.

Dal punto di vista del codice non vi sono modifiche alle funzioni “condivise” di UC1 per adattarle a UC2 in quanto Bundle risulta essere una estensione della classe Configurazione.



|  |  |
| --- | --- |
| **Nome del contratto** | **CO1: Info Configurazione** |
| **Operazione** | infoConfigurazione() |
| **Riferimenti** | Caso d’uso UC2: Crea Bundle acquistabile |
| **Pre-condizioni** | * È stata inizializzata un’istanza Conf di Configurazione |
| **Post-condizioni** | * È stata inizializzata un’istanza B di Bundle. * Gli attributi di B sono modificati sulla base dei parametri forniti. * L’attributo id di B è stato inizializzato con una sequenza unica. |

## UC3

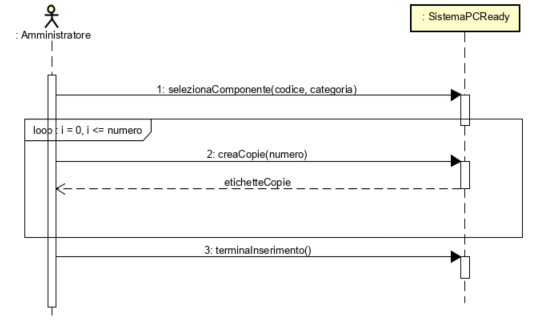


|  |  |
| --- | --- |
| **Nome del contratto** | **CO1: Crea Componente** |
| **Operazione** | creaComponente() |
| **Riferimenti** | Caso d’uso UC3: Inserisci Nuovo Componente |
| **Pre-condizioni** | * L’amministratore è autenticato |
| **Post-condizioni** | * È stata creata un’istanza C di Componente * L’istanza C di Componente è stata associata a Catalogo mediante l’associazione “È contenuto in” * Gli attributi di C sono stati inizializzati con i parametri forniti * È stata inizializzata una lista di CopiaComponente denominata “listaCopie” come attributo di C. |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO2: Crea Copie** |
| **Operazione** | creaCopie() |
| **Riferimenti** | Caso d’uso UC3: Inserisci Nuovo Componente |
| **Pre-condizioni** | * È in corso l’inserimento di un nuovo componente * È stata inizializzata una lista di CopiaComponente detta “listaCopie” |
| **Post-condizioni** | * Sono state create N istanze di Copia, dove N è parametro della funzione * Le N istanze di copia sono associate a C mediante l’associazione “Dispone di” * Un’etichetta è stata stampata per ogni copia * Il valore dell’attributo di C denominato “listaCopie” è stato aggiornato in base alle copie inizializzate |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO3: Termina Inserimento** |
| **Operazione** | terminaInserimento() |
| **Riferimenti** | Caso d’uso UC3: Inserisci Nuovo Componente |
| **Pre-condizioni** | * L’amministratore ha verificato la validità dell’inserimento delle copie del nuovo componente |
| **Post-condizioni** | * Il Sistema mantiene in memoria il Componente e le sue Copie appena generate |

## UC4

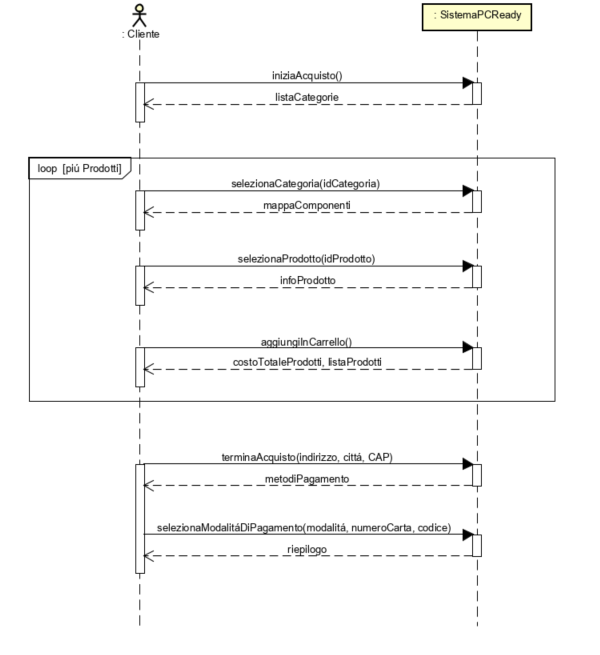


|  |  |
| --- | --- |
| **Nome del contratto** | **CO1: Seleziona Componente** |
| **Operazione** | selezionaComponente() |
| **Riferimenti** | Caso d’uso UC4: Inserisci Copia Componente |
| **Pre-condizioni** | * L’amministratore deve essere autenticato |
| **Post-condizioni** | * Viene recuperata l’istanza C dello specifico Componente cercato |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO2: Crea Copie** |
| **Operazione** | creaCopie() |
| **Riferimenti** | Caso d’uso UC4: Inserisci Copia Componente |
| **Pre-condizioni** | * È stata recuperata un’istanza C di Componente * È presente una lista di CopiaComponente detta “listaCopie” come attributo dell’istanza C |
| **Post-condizioni** | * Sono state create N istanze di Copia, dove N è parametro della funzione * Le N istanze di CopiaComponente sono associate a C mediante l’associazione “Dispone Di” * Un’etichetta è stata stampata per ogni copia * Il valore dell’attributo di C denominato “listaCopie” è stato aggiornato in base alle copie inizializzate |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO3: Termina Inserimento** |
| **Operazione** | terminaInserimento() |
| **Riferimenti** | Caso d’uso UC4: Inserisci Copia Componente |
| **Pre-condizioni** | * L’amministratore ha verificato la validità dell’inserimento delle copie del nuovo componente |
| **Post-condizioni** | * Il Sistema mantiene in memoria le N istanze di CopiaComponente appena generate |

## UC5



|  |  |
| --- | --- |
| **Nome del contratto** | **CO1: iniziaAcquisto** |
| **Operazione** | iniziaAcquisto() |
| **Riferimenti** | Caso d’uso UC5: Effettua Acquisto |
| **Pre-condizioni** | * Il cliente deve essere autenticato |
| **Post-condizioni** | * È stata inizializzata un’istanza di Carrello denominata “carrello” con i valori di default * “carrello” è stato associato ad Utente mediante la relazione “gestisce” * È stata inizializzata una lista “listaComponentiCarrello” di Componente, in quanto attributo di carrello |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO2: selezionaCategoria** |
| **Operazione** | selezionaCategoria() |
| **Riferimenti** | Caso d’uso UC5: Effettua Acquisto |
| **Pre-condizioni** | * Un’istanza di Carrello denominata “carrello” deve essere stata inizializzata * Il cliente ha visualizzato tutte le categorie di Componente |
| **Post-condizioni** | * Viene inizializzata una mappa idComponente-Componente denominata “mappaCorrente” * Il contenuto di mappaCorrente è stato aggiornato in base alla Categoria scelta |

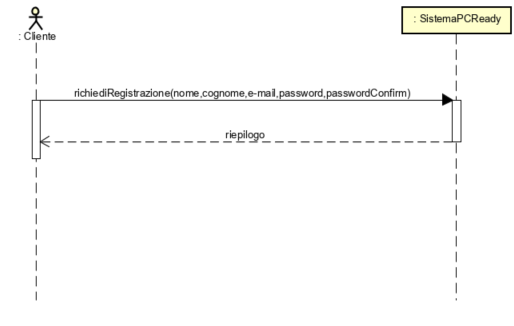
|  |  |
| --- | --- |
| **Nome del contratto** | **CO3: selezionaProdotto** |
| **Operazione** | selezionaProdotto() |
| **Riferimenti** | Caso d’uso UC5: Effettua Acquisto |
| **Pre-condizioni** | * È stata inizializzata una mappa idComponente-Componente denominata “mappaCorrente” * Un’istanza di Carrello denominata “carrello” è stata inizializzata |
| **Post-condizioni** | * È stata inizializzata un’istanza “componenteCorrente” di Componente |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO4: aggiungiInCarrello** |
| **Operazione** | aggiungiInCarrello() |
| **Riferimenti** | Caso d’uso UC5: Effettua Acquisto |
| **Pre-condizioni** | * È stata inizializzata un’istanza “componenteCorrente” di Componente |
| **Post-condizioni** | * L’attributo di “carrello” denominato “listaComponentiCarrello” è stato aggiornato con i valori dell’istanza “componente corrente”, mediante la relazione “contiene” tra Carrello e Componente * I restanti attributi di “carrello” sono stati aggiornati |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO5: terminaAcquisto** |
| **Operazione** | terminaAcquisto() |
| **Riferimenti** | Caso d’uso UC5: Effettua Acquisto |
| **Pre-condizioni** | * Il cliente ha selezionato tutti i componenti da acquistare |
| **Post-condizioni** | * È stata inizializzata un’istanza di Ordine denominata “ordineCorrente” * L’istanza “ordineCorrente” è stata associata a Cliente mediante la relazione “effettua” * Gli attributi di “ordineCorrente” sono stati modificati mediante i valori forniti dall’utente. * L’attributo di “ordine” denominato “listaComponentiOrdine” è stato aggiornato con i valori di “listaComponentiCarrello”. |

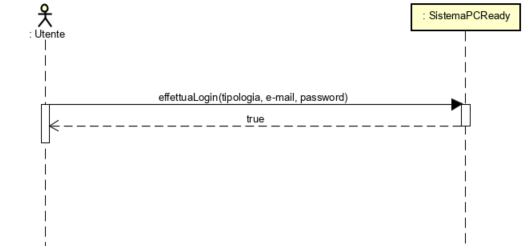
|  |  |
| --- | --- |
| **Nome del contratto** | **CO6: selezionaModalitáDiPagamento** |
| **Operazione** | selezionaModalitáDiPagamento() |
| **Riferimenti** | Caso d’uso UC5: Effettua Acquisto |
| **Pre-condizioni** | * Il cliente ha visualizzato i possibili metodi di pagamento |
| **Post-condizioni** | * Gli attributi di “ordineCorrente” sono stati modificati mediante i valori forniti dall’utente. * Il Sistema mantiene in memoria l’istanza “ordineCorrente” |

## UC6



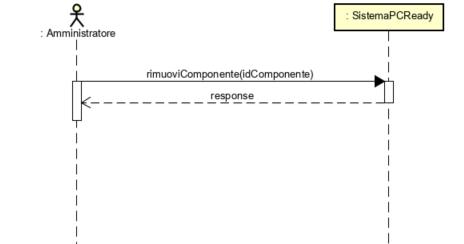
|  |  |
| --- | --- |
| **Nome del contratto** | **CO2: richiediRegistrazione** |
| **Operazione** | richiediRegistrazione() |
| **Riferimenti** | Caso d’uso UC6: Registrazione cliente |
| **Pre-condizioni** |  |
| **Post-condizioni** | * Viene inizializzata un’istanza di Cliente detta “clienteR” mediante l’associazione di Cliente con SistemaPCReady chiamata “registra” * Gli attributi di clienteR sono stati aggiornati mediante i dati forniti dall’utente * Il Sistema mantiene in memoria l’istanza “clienteR” |

## UC7



|  |  |
| --- | --- |
| **Nome del contratto** | **CO1: effettuaLogin** |
| **Operazione** | effettuaLogin() |
| **Riferimenti** | Caso d’uso UC7: Login |
| **Pre-condizioni** | * Esiste in memoria un’istanza di Utente o Amministratore che sta richiedendo il login |
| **Post-condizioni** | * È stata inizializzata un’istanza di Cliente “clienteCorrente” o di Amministratore “amministratoreCorrente”, in base alla corrispondenza trovata con i valori forniti. |

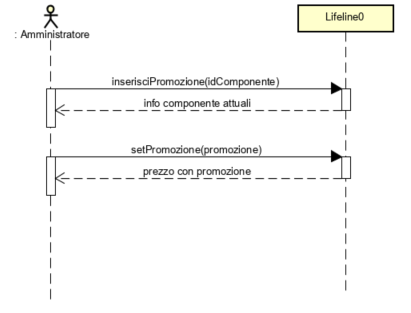
## UC8



|  |  |
| --- | --- |
| **Nome del contratto** | **CO1: rimuoviComponente** |
| **Operazione** | rimuoviComponente() |
| **Riferimenti** | Caso d’uso UC8: Rimozione componente |
| **Pre-condizioni** | * È stata inizializzata un’istanza “amministratoreCorrente” di classe “Amministratore”, ovvero un amministratore deve essere stato autenticato |
| **Post-condizioni** | * È stata eliminata un’istanza di Componente in base alla corrispondenza con l’identificativo fornito * È stata aggiornato l’attributo “mappaComponenti” della classe Catalogo |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO1: rimuoviConfigurazioni** |
| **Operazione** | rimuoviConfigurazioni() |
| **Riferimenti** | Caso d’uso UC8: Rimozione componente |
| **Pre-condizioni** | * È stata eliminata un’istanza di Componente di dato identificativo |
| **Post-condizioni** | * È stata eliminata un’istanza di Configurazione, il cui attributo “listaComponenti” conteneva una corrispondenza con l’istanza di Componente eliminata in precedenza * È stata eliminata un’istanza di Bundle, il cui attributo “listaComponenti” conteneva una corrispondenza con l’istanza di Componente eliminata in precedenza * È stata aggiornato l’attributo “mappaComponenti” della classe Catalogo |

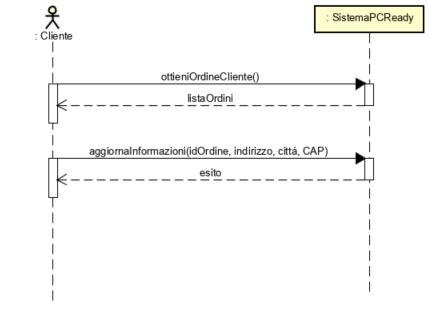
## UC9



|  |  |
| --- | --- |
| **Nome del contratto** | **CO1: selezionaComponente** |
| **Operazione** | selezionaComponente() |
| **Riferimenti** | Caso d’uso UC9: Crea promozione |
| **Pre-condizioni** | * È stata inizializzata un’istanza “amministratoreCorrente” di classe “Amministratore”, ovvero un amministratore deve essere stato autenticato |
| **Post-condizioni** | * È stata recuperata un’istanza di Componente “comp” in base alla corrispondenza con l’identificativo fornito |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO2: setPromozione** |
| **Operazione** | setPromozione() |
| **Riferimenti** | Caso d’uso UC9: Crea promozione |
| **Pre-condizioni** | * È stata recuperata un’istanza “comp” di Componente |
| **Post-condizioni** | * È stato aggiornato l’attributo “promozione” di “comp” in base ai dati forniti dall’amministratore |

## UC10



|  |  |
| --- | --- |
| **Nome del contratto** | **CO1: ottieniOrdineCliente** |
| **Operazione** | ottieniOrdineCliente() |
| **Riferimenti** | Caso d’uso UC10: Modifica ordine |
| **Pre-condizioni** | * È stata inizializzata un’istanza “clienteCorrente” di classe “Cliente”, ovvero un cliente deve essere stato autenticato |
| **Post-condizioni** | * È stata inizializzata una lista di “Ordine” detta “listaOrdini” in base all’attributo id di clienteCorrente |

|  |  |
| --- | --- |
| **Nome del contratto** | **CO2: aggiornaInformazioni** |
| **Operazione** | aggiornaInformazioni() |
| **Riferimenti** | Caso d’uso UC10: Modifica ordine |
| **Pre-condizioni** | * È stata inizializzata una lista di “Ordine” detta “listaOrdini” |
| **Post-condizioni** | * Gli attributi di un’istanza di ordine (individuata in base ai dati forniti dal cliente) sono stati modificati sulla base dei dati forniti, mediante l’associazione “Memorizza” presente tra le classi “SistemaPCReady” e “Ordine” |

# **Progettazione**

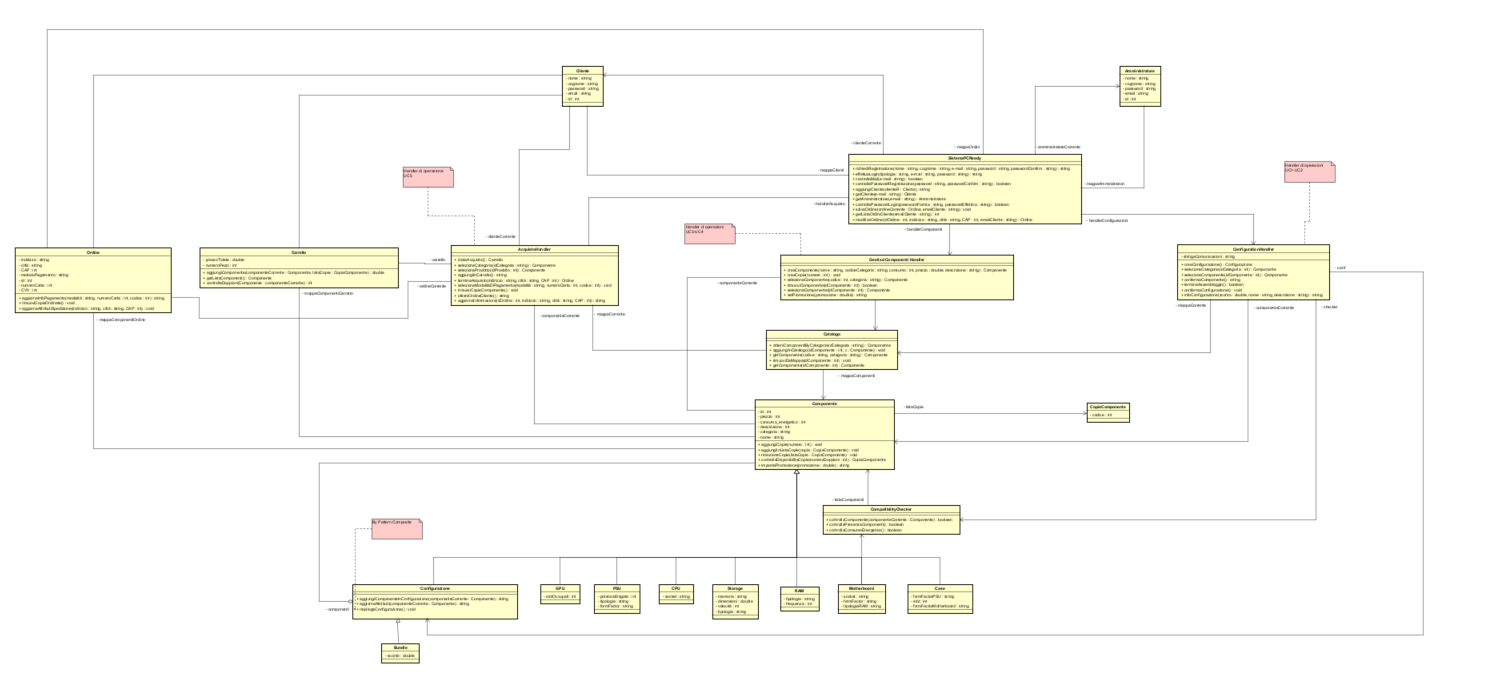
La progettazione orientata agli oggetti consente di definire e caratterizzare degli oggetti software, enfatizzando come questi collaborano per soddisfare tutti i requisiti individuati.

La progettazione è di fondamentale importanza, in quanto tutte le classi scelte vanno poi implementate durante la fase di programmazione orientata agli oggetti.

In tale capitolo verranno mostrati due elementi fondamentali per la progettazione:

* Diagramma delle classi di progetto: permette di descrivere il tipo degli oggetti che fanno parte di un sistema, marcando le tipologie di relazioni che vi sono tra di essi. [Punto di vista statico]
* Diagrammi di sequenza: mostrano l’interazione tra un insieme di oggetti documentando il comportamento dettagliato di un singolo scenario mediante un “formato a steccato”. [Punto di vista dinamico]

## Diagramma delle classi di progetto



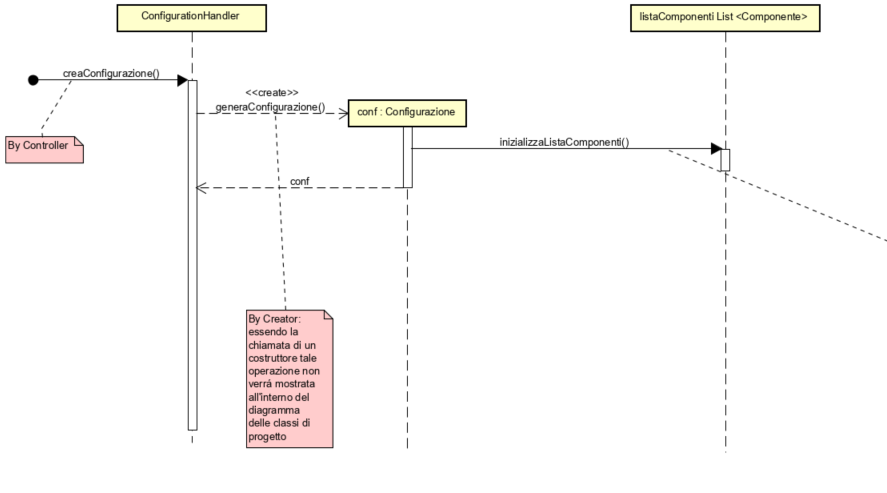
Si invita a visionare il file Astah in allegato alla documentazione per meglio apprezzare i dettagli di tale diagramma.

## Diagrammi di sequenza di sistema

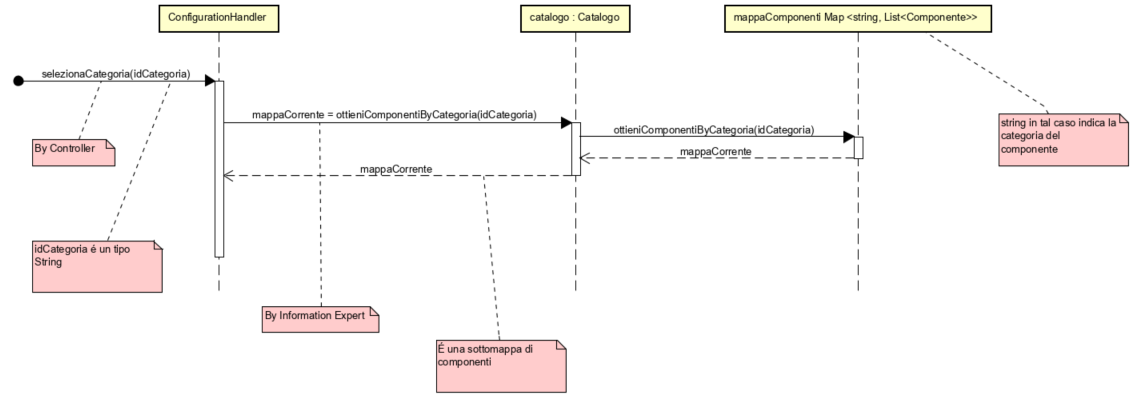
A seguito verranno mostrati i diagrammi di sequenza di sistema, raggruppati per casi d’uso: inoltre ciascun diagramma di sequenza di sistema verrà accompagnato da una breve descrizione

### SSD - UC1

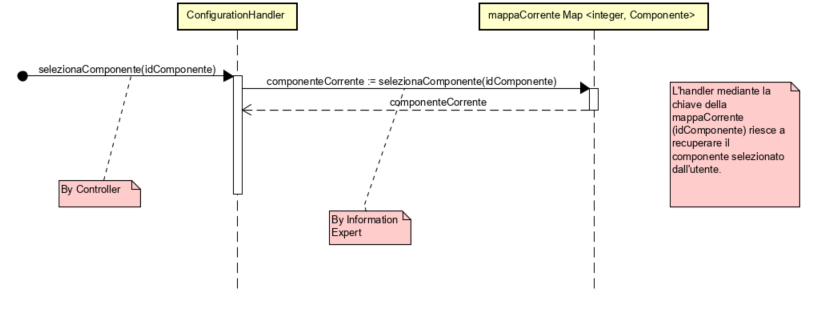
* **creaConfigurazione:** questa operazione si occupa di creare una configurazione, inizializzandone una listaComponenti, e impostando gli altri attributi ai valori di default.



* **selezionaCategoria:** tale funzione si occupa di prendere in ingresso la categoria selezionata da un utente, restituendo tutte le componenti contenute in tale categoria mediante una mappa con chiave intera (id del componente) e valore componente.

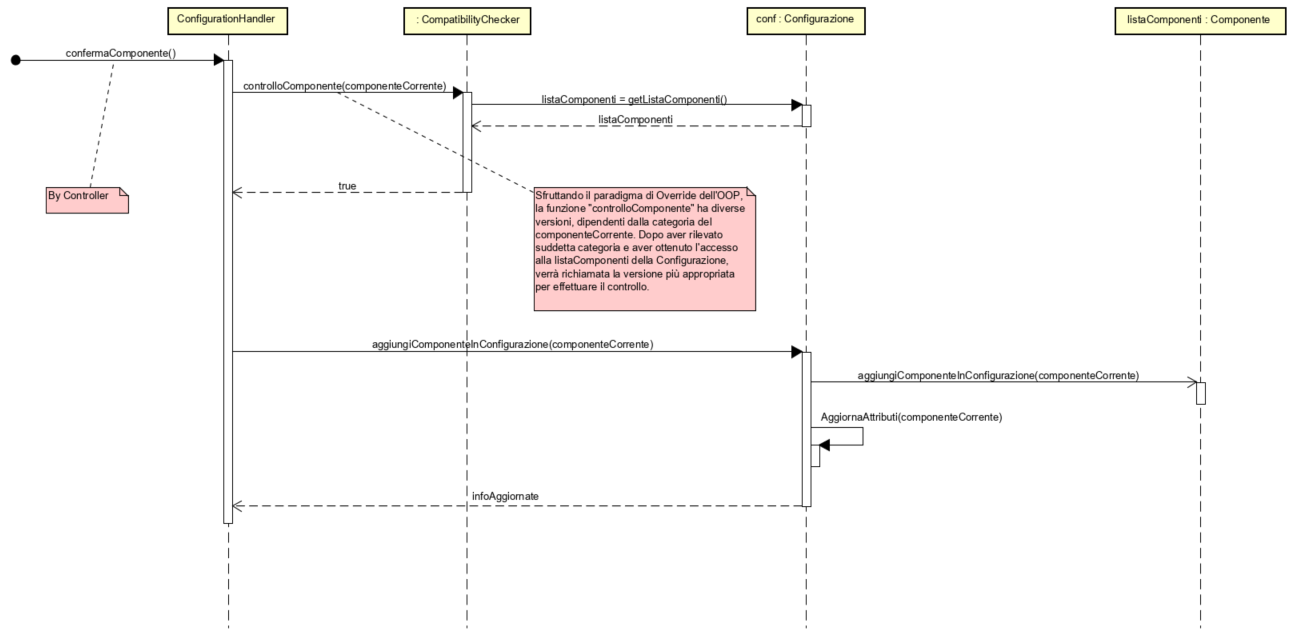


* **selezionaComponente:** tale funzione si occupa di restituire i dettagli del componente selezionato da un utente



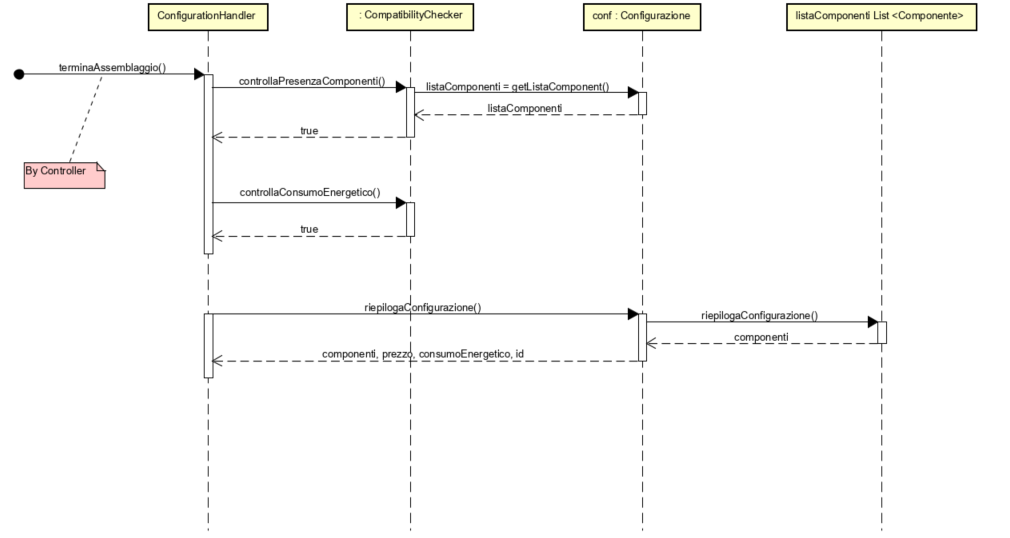
* **confermaComponente:** tale funzione permette di confermare il componente visionato in precedenza, e quindi di inserire quest’ultimo nella configurazione, se e soltanto se tale componente risulta essere compatibile con tutti i componenti già presenti nella “lista componenti” della configurazione.

Si osserva che la responsabilità dei controlli è stata affidata ad una classe apposita chiamata “CompatibilityChecker”.

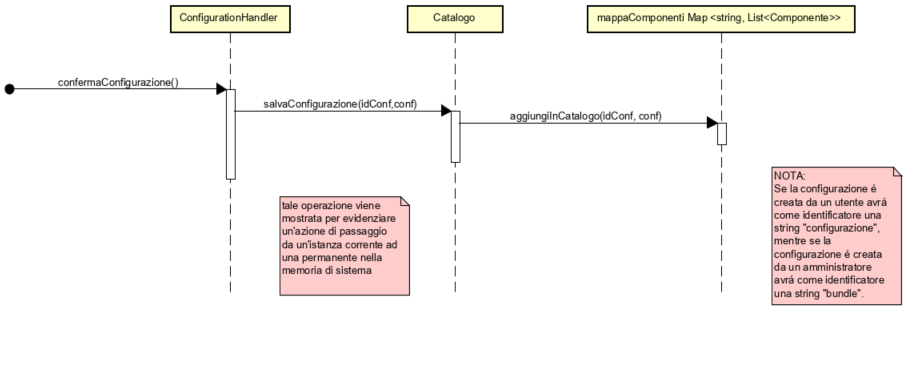


* **terminaAssemblaggio:** tale funzione consente di terminare l’assemblaggio della configurazione ed eseguire i controlli per il corretto funzionamento di quest’ultima.

Se i controlli vanno a buon fine verrà restituito un riepilogo della configurazione appena assemblata.

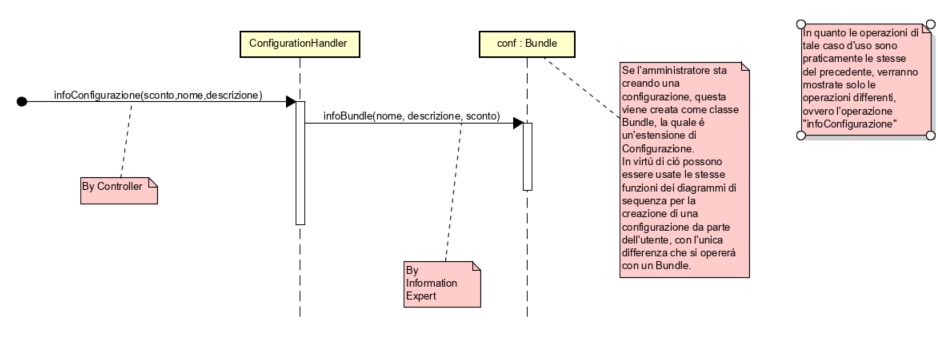


* **confermaConfigurazione:** tale funzione ha il compito di salvare nella memoria del sistema la configurazione appena creata.



### SSD – UC2

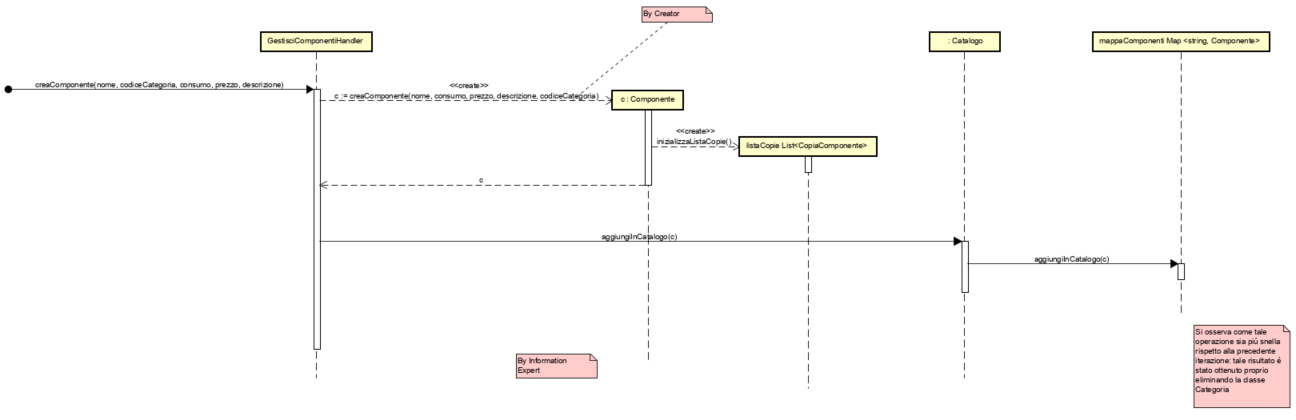
* **infoConfigurazione:** la funzione in questione ha il compito di impostare il nome, lo sconto e la descrizione del bundle che un amministratore ha assemblato.



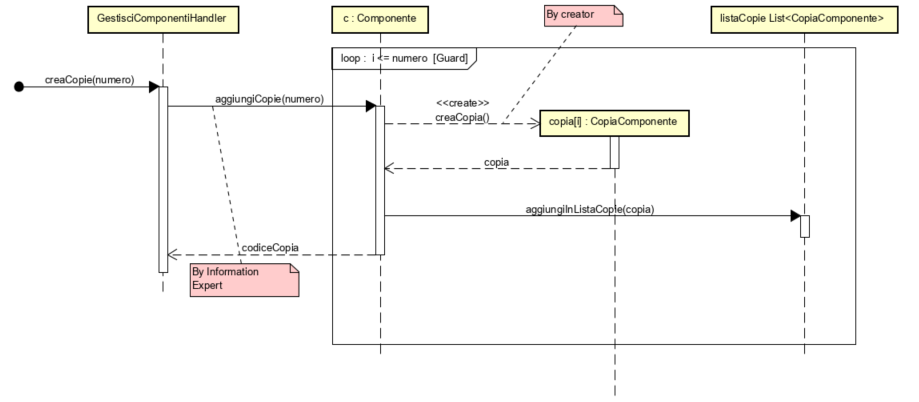
### SSD – UC3

* **creaComponente:** tale funzione ha il compito di creare un nuovo componente, prendendo in ingresso tutte le informazioni necessarie alla creazione.

Il componente creato verrá quindi aggiunto al catalogo.

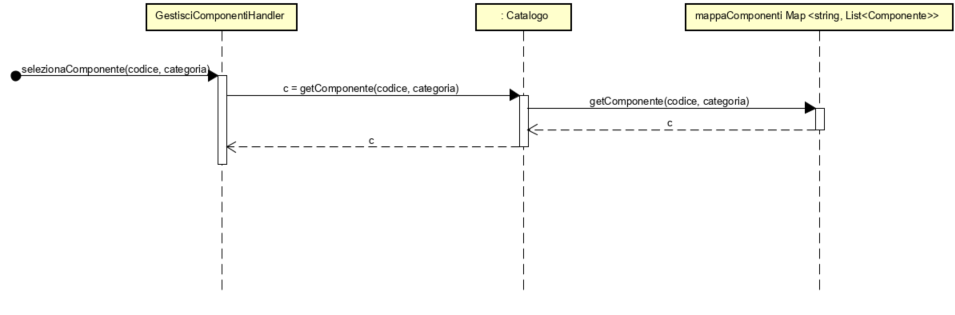


* **creaCopie:** la suddetta funzione ha il compito di inserire un determinato numero di copie per il componente [componenteCorrente] inserito mediante la funzione “creaComponente”.



### SSD – UC4

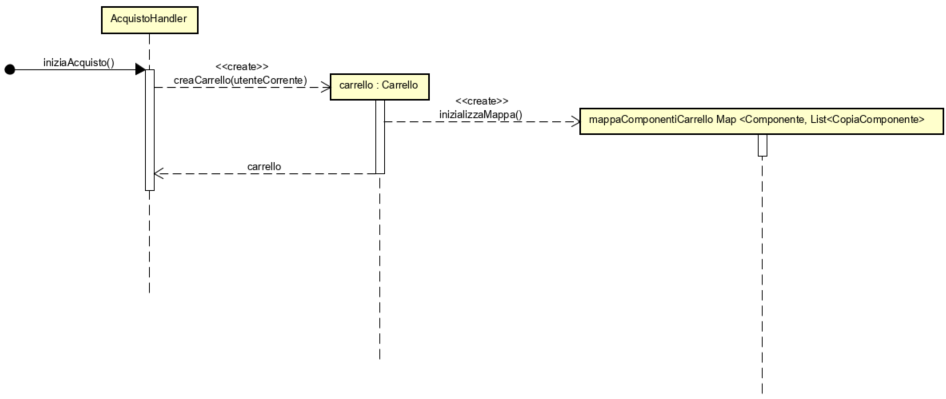
* **selezionaComponente:** tale funzione ha il compito di recuperare un componente già esistente in catalogo, per poter successivamente aggiungere delle copie di quest’ultimo mediante la funzione “creaCopie”.



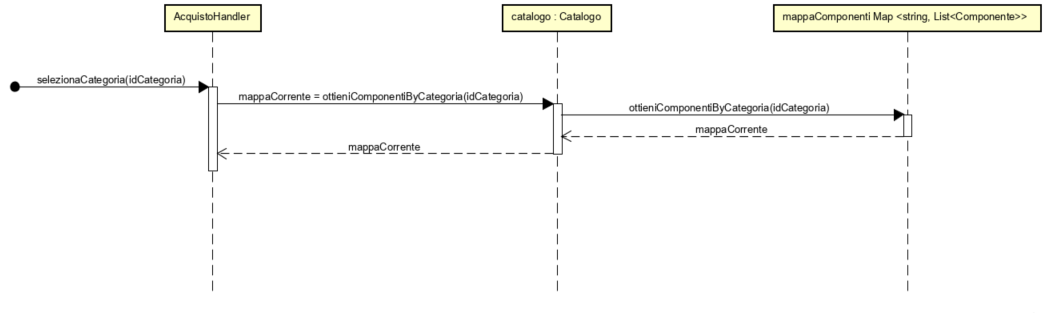
### SSD – UC5

* **iniziaAcquisto:** tale funzione ha il compito di inizializzare un “carrello” per l’utente che ha intenzione di effettuare un acquisto.

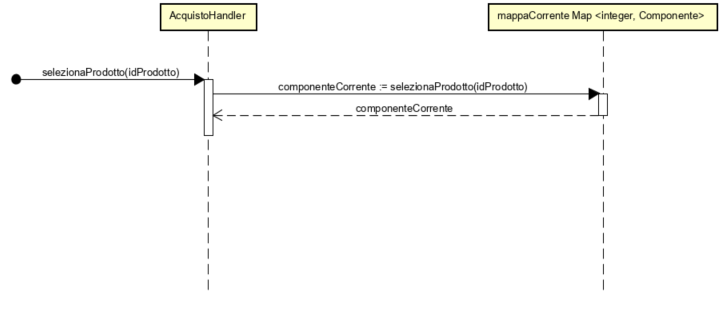
Il carrello in questione conterrà tutti i prodotti che l’utente ha intenzione di acquistare.



* **selezionaCategoria:** mediante tale funzione Viene selezionata una categoria dal cliente, e in base alla categoria scelta viene ritornata una mappa di id-componente/componente

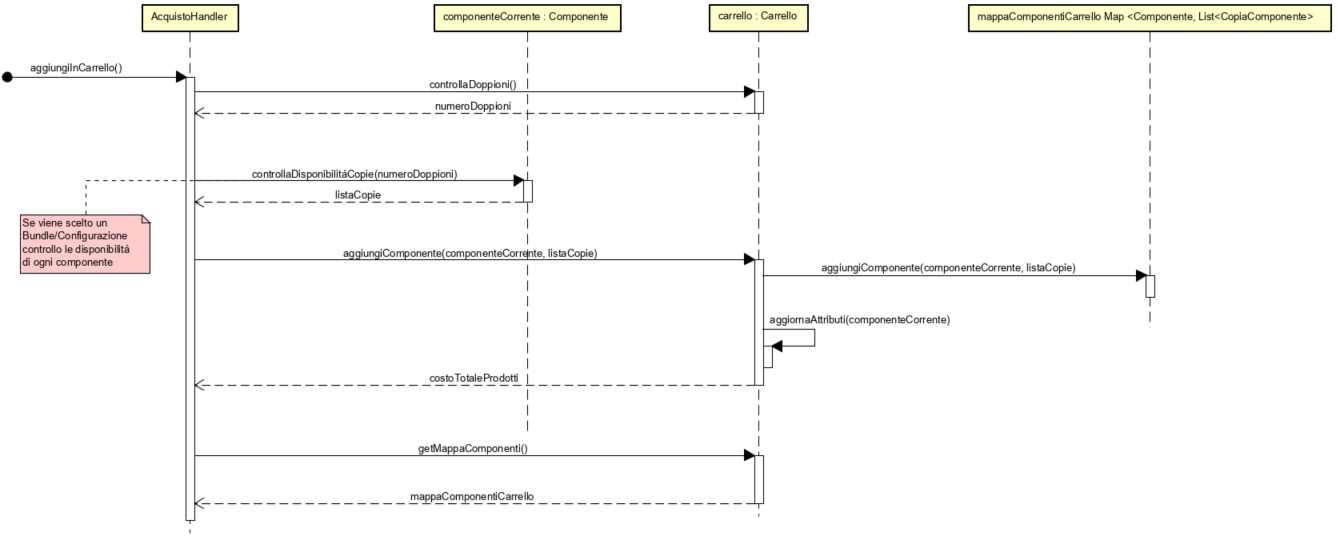


* **selezionaProdotto:** per mezzo di tale funzione viene selezionato dalla mappa ottenuta in precedenza (funzione selezionaCategoria) un prodotto, mediante l’id fornito dal cliente.

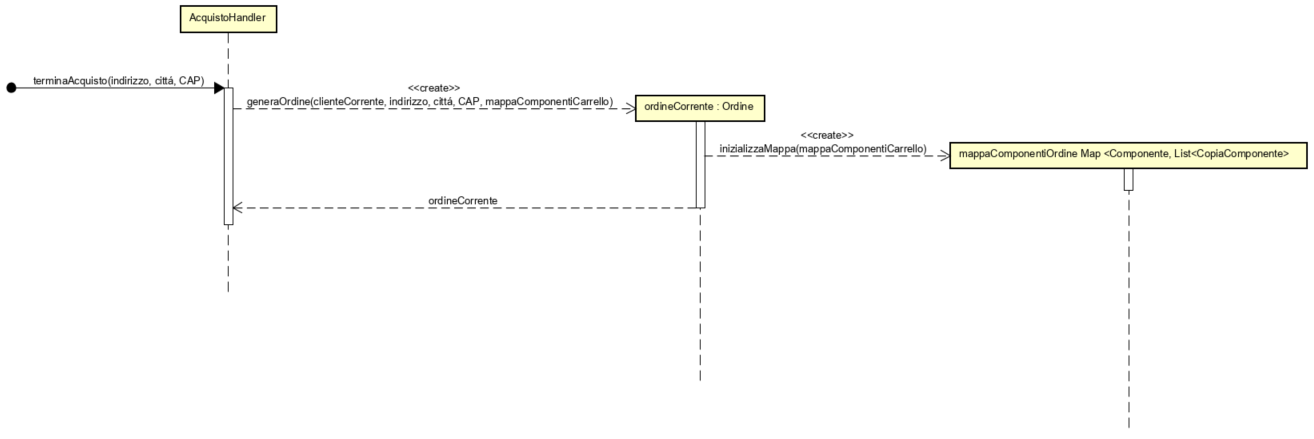


* **aggiungiInCarrello:** mediante tale funzione il prodotto selezionato in precedenza dal cliente viene aggiunto al carrello, dopo un controllo sulla disponibilità delle copie del componente

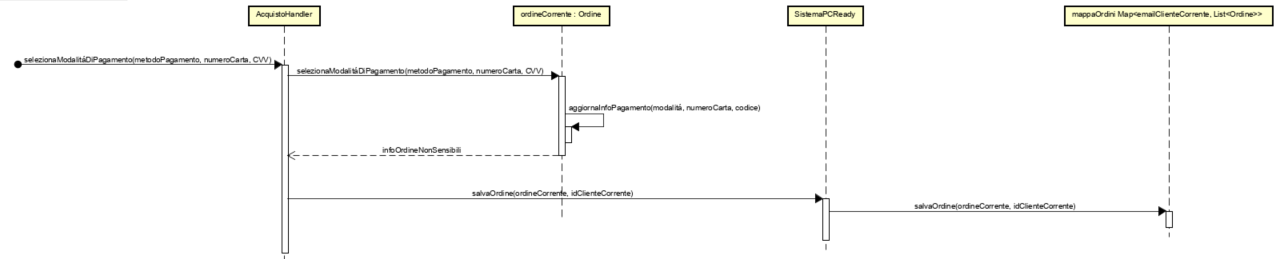
Per osservare in dettaglio la logica di controllo si consulti il documento “Discussione Iterazione 3”.



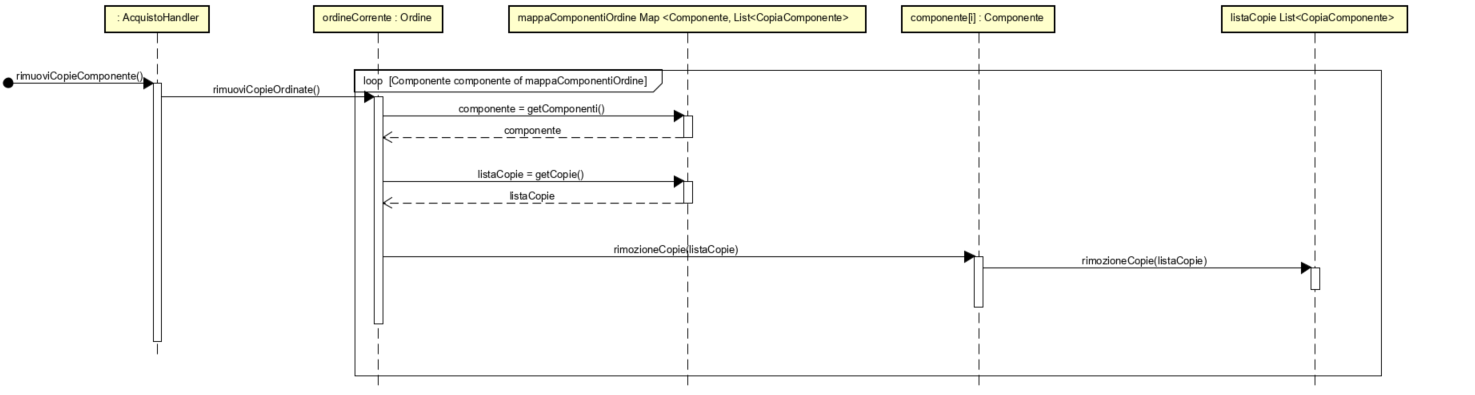
* **terminaAcquisto:** tale funzione consente di inizializzare un’istanza di ordine, la quale possiede una mappa di componenti che l’utente sta acquistando, associata ad un insieme di attributi relativi al domicilio del cliente.



* **selezionaModialitáDiPagamento:** Il cliente mediante questa funzione è in grado di inserire la modalità di pagamento preferita, e i dati della carta che desidera utilizzare per finalizzare l’acquisto. Inoltre l’ordine verrà salvato nella memoria del sistema.



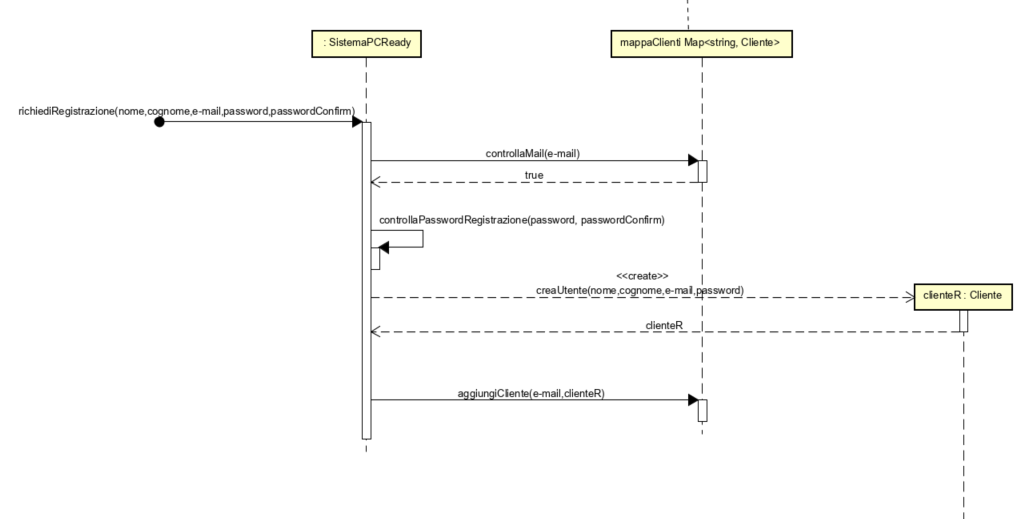
* **rimuoviCopieComponente:** In quanto l’acquisto è stato finalizzato, le copie che sono state acquistate dai clienti vanno rimosse in quanto non più disponibili. (L’id delle suddette copie viene comunque salvato nell’ordine effettuato, in modo tale da avere comunque traccia di ciò che è stato venduto).



### SSD – UC6

* **richiediRegistrazione:** tale funzione permette ad un utente di registrarsi al servizio PC Ready, fornendo al sistema una e-mail che non sia già in uso da un altro cliente.

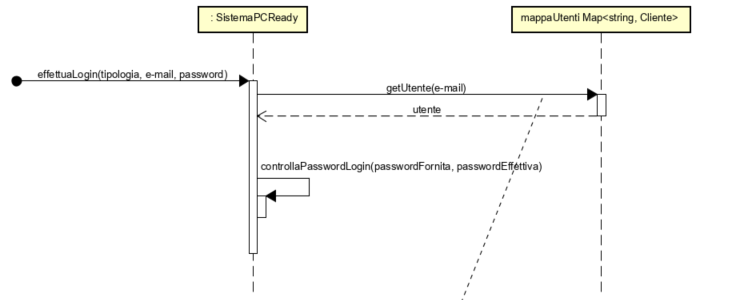
Viene eseguito anche un controllo sul campo password e conferma password, i quali devono essere coincidenti.

****

### SSD – UC7

* **effettuaLogin:** la funzione in questione consente ad un utente o ad un amministratore di effettuare il login.

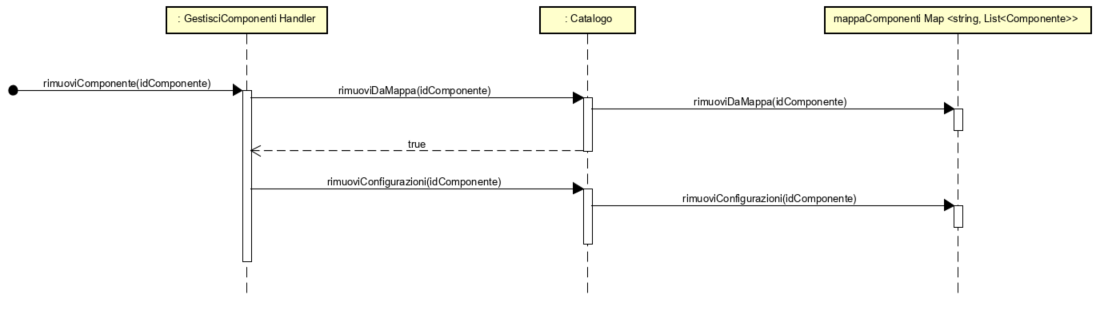
Il valore “tipologia” serve al sistema per comprendere se dovrà autenticare un amministratore o un cliente.



### SSD – UC8

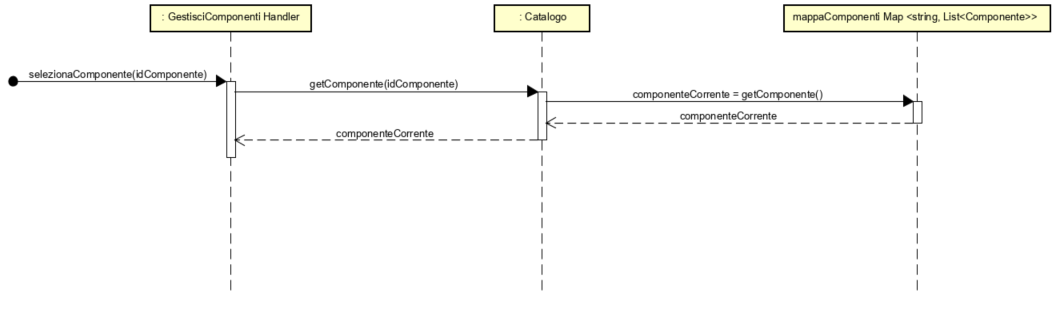
* **rimuoviComponente:** tale funzione permette di rimuovere un componente dal catalogo, mediante il suo identificativo.

Tale funzione prevede la rimozione di tutte le configurazioni e i bundle che contenevano il componente rimosso nella loro lista componenti.

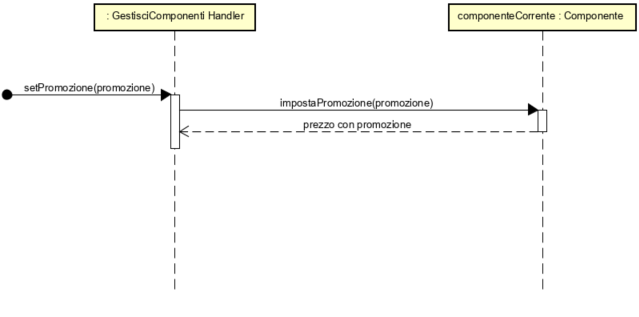


### SSD – UC9

* **selezionaComponente:** tale funzione permette di selezionare un componente dato il suo id dal catalogo, e consentire alla funzione successiva “setPromozione” di applicare una promozione al componente in questione.

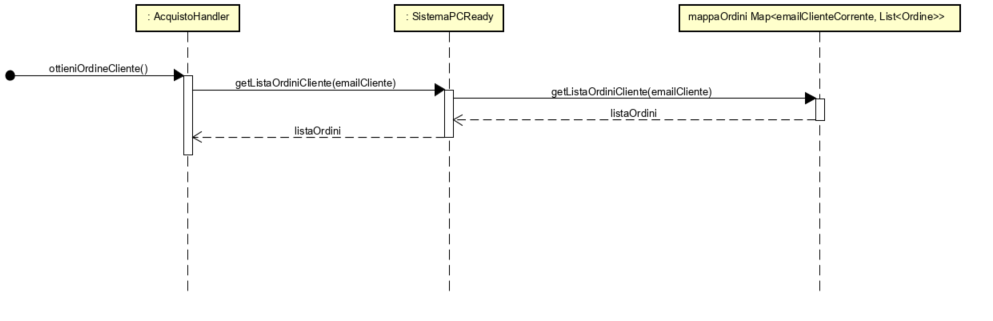


* **setPromozione:** tale funzione permette di variare l’attributo promozione di un componente scelto mediante la funzione precedente “selezionaComponente”.



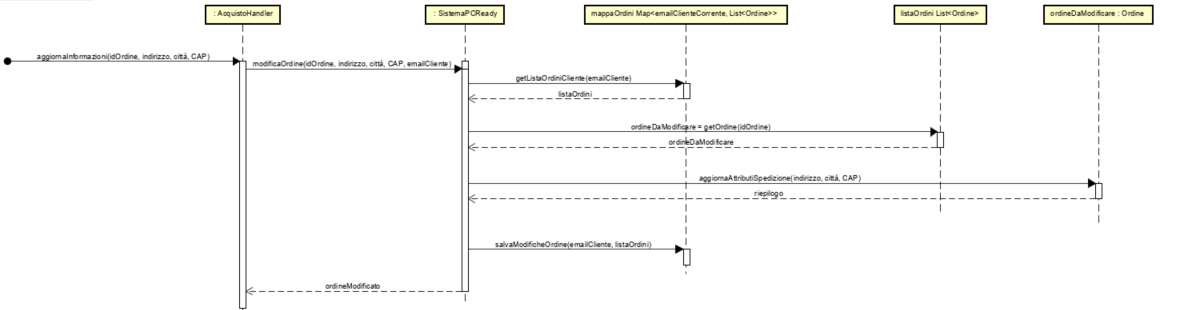
### SSD - UC10

* **ottieniOrdineCliente:** la funzione in questione consente di ottenere tutti gli ordini del cliente loggato, in modo da consentire al cliente di scegliere quale ordine modificare con la funzione successiva “aggiornaInformazioni”.



* **aggiornaInformazioni:** tale funzione permette di impostare delle nuove informazioni relative alla spedizione di un ordine, fornendo l’id dell’ordine da modificare e tutte le nuove informazioni necessarie.

Se viene fornito un id di un ordine non effettuato dal cliente corrente la funzione prevede un messaggio di errore.



# **Logica della fase di testing**

La fase di testing è stata di fondamentale importanza durante il processo di sviluppo di PCReady, in quanto ha permesso di rilevare, e successivamente correggere facilmente, dei malfunzionamenti e comportamenti anomali delle funzioni realizzate.

Un testing ben progettato consente di rilevare comportamenti indesiderati del software realizzato, e consente di eseguire delle operazioni di debugging ben mirate ai fini di correggere problemi logici e di progettazione del software.

Il nostro gruppo ha scelto di concentrarsi su dei testi di tipo unitario, ovvero dei semplici test che verificano la correttezza del codice, effettuati mediante JUnit5, ovvero mediante un framework di unit testing per il linguaggio di programmazione Java.

I nostri test di tipo unitario si sono focalizzati sulle funzioni proprie delle classi “core” del progetto, ovvero la classe PCReady, gli Handler di caso d’uso e la classe “CompatibilityChecker”.

A seguito di tali test, che saranno spiegati con un maggiore grado di dettaglio nei capitoli sottostanti, il gruppo è stato in grado di individuare errori di tipo “NullPointer” o eventuali operazioni non portate completamente a termine: tali problemi sono stati prontamente risolti dopo una breve analisi del codice.

## File AccessoTest

In tale file si è testato il funzionamento delle funzioni di login e registrazioni della classe “PC-Ready”.

Prima di eseguire i test descritti in seguito, viene eseguita una funzione di setup che genera delle mappe di clienti ed amministratori contenenti degli utenti di prova necessari per il controllo dei test, inoltre viene anche recuperata l’unica istanza del sistema PC-Ready presente.

**Login Test:** In questo test è stato visionato il corretto funzionamento della funzione effettuaLogin() del sistema PC-Ready, la quale prende in ingresso l’email dell’utente, la password e soprattutto la tipologia con il quale l’utente specifica se sta effettuando l’accesso come cliente o come amministratore. Dopo aver specificato chi sta effettuando l’accesso, tramite uno switch-case, effettuerà una distinzione in due casi. Nel primo caso ad effettuare l’accesso è un amministratore, in questo caso quest’ultimo verrà ricercato all’interno della mappa degli amministratori memorizzata dal sistema. Nel caso dell’utente fa la stessa operazione, ma la ricerca avverrà nella mappa dei clienti memorizzata. Per effettuare il testing in esame, le mappe clienti ed amministratori sono state generate in maniera provvisoria, e passate al sistema, il quale tramite la funzione effettuaLogin(), precedentemente indicata, ha controllato la presenza di un utente, anch’esso generato per effettuare il test, all’interno della mappa. Il controllo del funzionamento della funzione è stato effettuato con il comando, fornito da JUNIT5, assertNotEquals(), con il quale sono stati paragonati il messaggio restituito dalla funzione effettuaLogin() e il messaggio di default che la stessa funzione restituisce nel caso in cui il login vada a termine con esito negativo.

**Registrazione Test:** In questo test è stato visionato il corretto funzionamento della funzione richiediRegistrazione() del sistema PC-Ready, la quale prende in ingresso nome, cognome e l’email dell’utente oltre che alla sua password e al conferma\_Password necessario per assicurarsene il corretto inserimento. Dopo aver specificato tali informazioni, la funzione genererà un nuovo cliente con questi attributi e lo inserirà all’interno della mappa dei clienti memorizzata dal sistema. Per effettuare il testing in esame viene per prima cosa generato un nuovo cliente ed inserito nella mappa dei clienti tramite la funzione specificata in precedenza, e si controlla che l’inserimento è andato a buon fine attraverso il comando, fornito da JUNIT5, assertNotNull(), con il quale controlliamo che, all’interno della mappa è presente l’utente appena inserito. Poiché la funzione richiediRegistrazione() impedisce l’inserimento di un nuovo utente con un email già presente nella mappa, viene effettuato un controllo tramite il comando, fornito da JUNIT5, assertEquals(), con il quale controlliamo che in risposta alla nuova richiesta di inserimento è stato restituito il messaggio di errore di default.

## File AcquistoHandlerTest

In tale file si è testato il funzionamento della classe “AcquistoHandler”, la quale gestisce tutte le operazioni dei casi d’uso UC5, ovvero si occupa di tutte le operazioni di creazione di un nuovo carrello e di un ordine che permetterà poi di concludere l’acquisto di un componente o di una configurazione/bundle.

Prima di eseguire i test descritti in seguito, viene eseguita una funzione di setup che genera un cliente, generato appositamente per il test, necessario per potergli poi associare un carrello ed un ordine.

**Aggiungi al carrello Test:** In questo test è stato visionato il corretto funzionamento della funzione aggiungiInCarrello() dell’AcquistoHandler, la quale ha il compito di inserire una nuova componente o configurazione all’interno del carrello di un utente. Tale funzione può avere due sviluppi diversi, distinte tramite uno switch-case. Nel primo caso, in cui viene inserita nel carrello una nuova configurazione, essa viene aggiunta insieme alle copie dei singoli componenti che la costituiscono, controllandone la disponibilità. Nel secondo caso, che si ha con l’aggiunta di una singola componente, il procedimento è lo stesso del primo sono che il controllo e l’aggiunta della copia viene fatta per un singolo componente. L’obiettivo del testing è proprio quello di controllare che tali funzioni di inserimento nel carrello e controllo del numero di copie funzionino correttamente. Tale controllo viene effettuato generando un nuovo carrello e inserendo in esso tre copie di uno stesso componente di cui però ne sono disponibili solo due nel catalogo. Tramite il comando, fornito da JUNIT5, assertTrue(), viene controllato proprio che il numero di componenti inserite nel carrello è di due controllando quindi che sia l’inserimento che il controllo sul numero delle copie sia andato a buon fine.

**Aggiungi configurazione Test:** Questo test ci permette di controllare il corretto funzionamento dell’inserimento nel carrello di una configurazione, inoltre tale inserimento viene accompagnato da un ulteriore doppio inserimento di una componente “CPU” per controllare, anche in questo testing, il corretto funzionamento della gestione dei doppioni. La funzione dell’AcquistoHandler interessata in questo test è sempre aggiungiInCarrello() che già abbiamo analizzato nella descrizione precedente. Il testing si basa sulla generazione di una configurazione partendo dalla generazione delle sue componenti, fatto ciò la configurazione viene inserita all’interno del carrello insieme alle due componenti uguali di cui però si ha una sola copia disponibile. Per controllare che tutto è andato a buon fine, tramite il comando, fornito da JUNIT5, assertTrue() viene controllato che le componenti inserite nel carrello sono effettivamente due, la configurazione e una sola CPU.

**Ordine Test:** In questo test è stato visionato il corretto funzionamento della funzione terminaAcquisto() dell’AcquistoHandler, la quale si occupa di generare, prendendo in ingresso indirizzo, città e CAP della spedizione, un nuovo ordine, per le componenti inserite nel carrello di un cliente corrente. per effettuare il testing di tale funzione sono stati passati in ingresso dei dati casuali per terminare l’ordine e successivamente tramite il comando, fornito da JUNIT5, assertEquals(), è stato controllato che l’ordine è stato effettivamente generato paragonando la citta dell’ordine creato con la città che è stata scelta durante la creazione di quest’ultimo. Poiché i risultati coincidevano abbiamo avuto la conferma della creazione di un ordine destinato a quella città.

**Ottieni ordine Test:** In questo test è stato visionato il corretto funzionamento della funzione ottieniOrdineCliente() dell’AcquistoHandler, la quale si occupadi elencare tutti gli ordini di un cliente avendo accesso alla sua lista degli ordini data la sua email. Tale funzione si basa sulla funzione getListaOrdiniCliente() che si occupa di ottenere, sempre tramite la mail, la lista completa degli ordini. Nel testing, tramite il comando, fornito da JUNIT5, assertEquals(), viene paragonata proprio la lista ordini di un cliente, di cui si ha piena conoscenza, con la lista ottenuta proprio con la funzione ottieniOrdineCliente(). Poiche il comando assertEquals() ci da esito positivo, ne deduciamo che le due liste sono uguali.

**Aggiorna ordine Test:** In questo test è stato visionato il corretto funzionamento della funzione aggiornaInformazioni() dell’AcquistoHandler, la quale si occupadi modificare le informazioni su di un suo ordine prendendo in ingresso i dati da modificare e l’id dell’ordine in cui devono essere cambiati. Viene sfruttata la funzione del sistema PCReady modificaOrdine() che permette proprio di cambiare le informazioni di un ordine. Per effettuare il test abbiamo usato il comando, fornito da JUNIT5, assertNotEquals(), per paragonare un ordine di un determinato cliente e lo stesso ordine dopo aver effettuato le modifiche dei suoi dati proprio tramite la funzione presa in esame. Poiché il comando assertNotEquals() da esito positivo, ne deduciamo che l’ordine è stato modificato correttamente.

## File CompatibilityCheckerTest

In tale file si è testato il funzionamento della classe CompatibilityChecker, la quale ha il compito di verificare la compatibilità tra le varie componenti inserite nella configurazione.

In quanto le logiche di controllo compatibilità, nonostante le diversità tra le componenti, sono molto similari tra loro, si è deciso di testare solo i controlli di maggior impatto nell’assemblaggio di una configurazione o bundle.

Dunque, ciascuna delle funzioni sotto riportate, testa uno specifico override della funzione “controlloComponente(Componente comp)”: tali override prendono in ingresso un componente specializzato e la lista delle componenti attuali della configurazione, ai fini di poter eseguire un controllo con tutte le componenti attualmente presenti della configurazione.

**CPU Incompatibile Test:** tale test prevede, una volta inizializzato l’handler Configurazioni nel setup, l’inserimento nella configurazione di una Motherboard e di una CPU incompatibile con la suddetta motherboard, per poi provare l’inserimento di una CPU compatibile. Dunque, mediante una funzione “assertEquals()” si controlla se il secondo componente della lista della configurazione risulta essere proprio la CPU compatibile che si è provato ad inserire.

**Motherboard Incompatibile Test:** tale test, dopo aver rimosso la motherboard precedente, mantiene una CPU di socket “AM4” nella configurazione, e prova ad inserire una motherboard di socket “FCLGA1151”. Dunque si re-inserisce la motherboard con socket “AM4” e si controlla mediante la funzione “assertEquals()” se il secondo componente della lista della configurazione risulta essere proprio la Motherboard compatibile con socket “AM4” che si è provato ad inserire.

**Gpu Incompatibile Test:** tale test prevede un inserimento in una configurazione, che già contiene una motherboard e una CPU, di un Case con 4 slot disponibili e di due GPU: la prima occupa 2 slot case, mentre la seconda occupa 6 slot (ovviamente non esiste una tale GPU standard in mercato, si esagera solo per testare la funzione di controllo… ). Mediante una funzione “assertTrue()” si controlla, mediante un contatore, che la configurazione attuale contenga solo una GPU, ovvero solo la GPU che occupa 2 slot case.

## File ConfigurationHanlderTest

In tale file si è testato il funzionamento della classe “ConfigurationHandler”, la quale gestisce tutte le operazioni dei casi d’uso UC1 e UC2, ovvero si occupa di tutte le operazioni di creazione di una configurazione o di un bundle.

In quanto la creazione di una configurazione sfrutta le stesse funzioni della creazione di un Bundle (a meno della funzione “infoConfigurazione”), i primi 4 test riguardano la creazione di una configurazione.

Prima di eseguire i test descritti in seguito, viene eseguita una funzione di setup che inserisce dei componenti manualmente all’interno del catalogo utilizzato dal configuration handler.

**Seleziona Categoria Test:** tale test controlla il corretto funzionamento della funzione “selezionaCategoria”, e controlla mediante una funzione “assertNotNull” che la mappa restituita dalla funzione “selezionaCategoria” non sia nullo.

**Seleziona Componente Test:** tale test controlla il corretto funzionamento della funzione “selezionaComponente”, difatti si cerca di ottenere una RAM id pari a 3. Mediante una funzione “assertNotNull” si verifica che il componente restituito dalla funzione “selezionaComponente” non sia nullo.

**Conferma Componente Test:** tale test controlla il corretto funzionamento della funzione “confermaComponente”, cercando di inserire nella configurazione corrente la RAM precedentemente selezionata. Mediante una funzione “assertTrue” si controlla che il primo componente della lista di componenti della configurazione attuale sia effettivamente di categoria “RAM”.

**Conferma Termine Assemblaggio Test: :** tale test controlla il corretto funzionamento della funzione “terminaAssemblaggio” e della funzione “confermaConfigurazione”, dopo aver correttamente inserito un set di componenti compatibili nella lista componenti di una configurazione. Mediante una funzione “assertEquals” si controlla che la configurazione appena creata abbia una lista di componenti uguale alle componenti appena inserite.

**Crea Bundle Test:** tale test controlla il corretto funzionamento della funzione “terminaAssemblaggio”, della funzione “infoConfigurazione” e della funzione “confermaConfigurazione”, nel caso di creazione e di inserimento di un bundle nel catalogo. Mediante una funzione “assertTrue” si controlla che il numero dei Bundle contenuti sia effettivamente pari ad 1, ovvero che sia stato effettivamente inserito il bundle appena creato.

**Doppia Motherboard Test:** tale test controlla che la funzione “terminaAssemblaggio” impedisca effettivamente l’inserimento di un componente tale da essere unico nella configurazione, provando ad inserire una seconda motherboard nella configurazione. Nonostante tale operazione sia di responsabilità della classe “compatibilityChecker” si vuole effettivamente controllare se vi sono errori dovuti alla gestione del risultato dei controlli della funzione “terminaAssemblaggio”. Mediante una funzione “assertTrue” che prende in ingresso un contatore, si controlla che la configurazione che si sta creando contenga effettivamente 1 e una sola motherboard.

**Componente Incompatibile Test:** tale test controlla che la funzione “terminaAssemblaggio” impedisca effettivamente l’inserimento di un PSU tale da non fornire abbasta potenza per i restanti componenti selezionati per la configuazione. Nonostante tale operazione sia di responsabilità della classe “compatibilityChecker” si vuole effettivamente controllare se vi sono errori dovuti alla gestione del risultato dei controlli della funzione “terminaAssemblaggio”, cosí come si è visto per il precedente test. Per tale motivo si elimina il PSU precedente e lo si sostituisce con un PSU che fornisce solo 50W, e lo si inserisce in configurazione. Mediante una funzione “assertFalse” si controlla che la funzione “terminaAssemblaggio” ritorni false, dovuto al fatto che il PSU non è compatibile con i componenti scelti.

## File GestisciComponentiHandlerTest

In tale file si è testato il funzionamento della classe GestisciComponentiHandler, la quale ha il compito gestire la creazione, la selezione e la rimozione di una determinata componente. Si occupa anche della creazione di nuove copie di una specifica componente e della generazione di promozioni da applicare.

**Seleziona Componente Test:** In questo test è stato visionato il corretto funzionamento della funzione selezionaComponente() di GestisciComponenteHandler, la quale si occupa semplicemente di estrarre una componente dal catalogo dato il suo id. Nel Test controlliamo proprio che, dato un componente che siamo certi essere presente all’interno del catalogo, questo venga trovato e restituito dalla funzione presa in esame. Tale controllo è stato effettuato tramite il comando, fornito da JUNIT5, assertNotNull(), che ci permette di controllare se è stata effettivamente trovata nel catalogo una componente corrispondente a quella cercata.

**Crea Componente Test:** In questo test è stato visionato il corretto funzionamento della funzione creaComponente() di GestisciComponenteHandler, che, prendendo in ingresso i dati necessari, dipendenti da che tipo di componente si vuole creare, si occupa di generare la componente richiesta e di aggiungerla al catalogo. Nel Test controlliamo proprio che il componente che creiamo noi appositamente nel test venga inserito correttamente nel catalogo. Per fare ciò sfruttiamo la funzione selezionaComponente() del handler, di cui abbiamo testato il funzionamento in precedenza, per controllare, tramite il comando, fornito da JUNIT5, assertNotNull(), che nel catalogo, dopo la creazione, è presente l’elemento appena generato.

**Crea Copie Test:** In questo test è stato visionato il corretto funzionamento della funzione creaCopia() di GestisciComponenteHandler, la quale si occupa di generare, dato un numero n in ingresso, una quantità pari a tale numero di copie di un determinato componente. Le copie vengono poi inserite all’interno di una lista che conterrà tutte le copie di quel componente. Nel Test controlliamo proprio che queste n copie vengano effettivamente generate. Per fare ciò sfruttiamo il comando, fornito da JUNIT5, assertNotEquals(), con il quale paragoniamo la dimensione della lista di componenti di un componente corrente prima e dopo l’inserimento. Poiché il test da esito positivo abbiamo stabilito che le dimensioni delle due liste sono diverse e che ciò è dovuto all’inserimento in essa delle nuove copie.

**Set Promozione Test:** In questo test è stato visionato il corretto funzionamento della funzione setPromozione() di GestisciComponenteHandler, la quale ha il compito di impostare una percentuale di sconto ad un determinato componente. Nel test controlliamo proprio che la promozione viene impostata correttamente e per fare ciò, preso un componente generato appositamente per il test, controlliamo, tramite il comando, fornito da JUNIT5, assertNotEquals(), che il valore del suo attributo promozione, memorizzato in due variabili prima e dopo aver utilizzato la funzione setPromozione(), risulti essere cambiato, facciamo ciò proprio paragonando le due variabili appena create.

**Rimuovi Componente Test:** In questo test è stato visionato il corretto funzionamento della funzione rimuoviComponente() di GestisciComponenteHandler, che ha il compito di rimuovere una componente dal catalogo dandogli in ingresso l’id del componente da rimuovere. Per effettuare tale test abbiamo sfruttato il comando, fornito da JUNIT5, assertNull(), con il quale abbiamo controllato che all’interno del catalogo, dopo la rimozione di una componente, di cui eravamo certi della sua presenza nel catalogo, tale componente non fosse più presente.

# **Persistenza dei dati: logica del Parser**

Nell’ambito del problema della permanenza dei dati, abbiamo analizzato prima di tutto la complessità che un sistema di tipo Database dovrebbe rispettare per meglio rappresentare gli oggetti creati e manipolati dal sistema PCReady, come tale complessità potrebbe crescere al progredire delle varie iterazioni del processo software, e una stima del tempo necessario per realizzarlo e farlo evolvere di pari passo con il resto del software.

A conclusione delle nostre ricerche è stato valutato che la priorità assoluta vada all’obbiettivo di mostrare al cliente un software funzionante e dal numero di funzioni in costante aumento, e che quindi la costruzione di un database ben strutturato e articolato – seppur di rilevante importanza – possa tagliare fuori una parte importante del tempo a nostra disposizione fra un’iterazione e la successiva.

Un’ulteriore indagine ci ha portato alla conclusione che sia possibile commissionare il lavoro sul Database a terzi (in subappalto ad un’azienda o assumendo un “database engineer”), in modo tale da permettere al nostro team di sviluppatori di limitare la mole di lavoro allo sviluppo di un’interfaccia fra sistema PCReady e database.

Per lo sviluppo di tale interfaccia, si è presupposto che il Database fornirà un server costruito su approccio REST, e che quindi l’accesso ai dati o il loro salvataggio prenda la forma di una richiesta HTTP contenente un file di testo in formato JSON.

Suddetta interfaccia è quindi ciò a cui ci siamo riferiti nel resto della documentazione con il termine “Parser”, in quanto si occupa di analizzare e fare da tramite fra oggetti in memoria e oggetti testuali, e la sua implementazione è presente nell’omonima classe.

## Il database d’esempio

In assenza dell’esistenza fisica del server collegato al Database, si è deciso – senza particolari restrizioni – di poter operare in maniera perfettamente analoga su uno pseudo-database formato da tre file in formato JSON, i quali possono simulare perfettamente il contenuto di un’eventuale richiesta HTTP al server effettivo. I file sono contenuti nella cartella **“data”**, esterna al source *“src”*.

Tali file sono, in ordine alfabetico: ***catalogo.json*** , ***ordini.json*** , ***utenti.json***.

**Catalogo:** il file *catalogo.json* rappresenta la permanenza in memoria di un oggetto appartenente alla classe Catalogo. In quanto tale, per rappresentare l’elenco dei componenti come una Mappa di Liste, il file rappresenta un singolo oggetto i cui attributi sono i nomi delle varie tipologie di Componente esistenti (“Storage”, “CPU”, etc.) e i valori corrispondenti sono degli array.

Tali array, a loro volta, rappresenteranno le Liste di Componenti di quella specifica categoria, e pertanto contengono oggetti contenenti tutti gli attributi necessari per istanziare un Componente (id, nome, descrizione, prezzo, consumo), più gli attributi aggiuntivi specifici di quella sotto-classe (i.e. la “potenza erogata” per i PSU).

Inoltre, ciascun oggetto JSON di Componente contiene anche un array, il cui contenuto sono i codici numerici delle sue CopieComponente presenti.

Configurazioni e Bundle costituiscono un’eccezione particolare, in quanto per essere istanziati richiedono l’esistenza apriori degli altri Componenti. Di conseguenza, questi due attributi saranno processati dal Parser per ultimi. Il contenuto di tali oggetti si limita al campo numerico “id” e ad un array di numeri interi rappresentanti gli “id” dei Componenti che costituiscono la Configurazione/il Bundle: essi possono essere usati per ottenere le effettive istanze di Componente interrogando il Catalogo parziale, ottenuto processando il resto del file.

**Utenti:** il file *utenti.json* rappresenta un oggetto con soli due attributi: “Clienti” e “Amministratori”. In corrispondenza di ciascuna delle due etichette è presente una lista di oggetti, i quali verranno utilizzati per creare – rispettivamente – un Cliente o un Amministratore.

In ciascuno dei due casi, l’oggetto conterrà i 5 campi necessari per istanziare l’oggetto in memoria: id, nome, cognome, email, password.

**Ordini:** il file *ordini.json* deve esplicitamente essere richiesto per ultimo, in quanto dipende dalla conoscenza apriori del Catalogo e della lista dei Clienti.

Tale fila contiene un array di oggetti, ciascuno rappresentante un Ordine a sé stante. Per far ciò, l’oggetto contiene un campo “cliente”, il cui contenuto equivale ad una email valida per poter accedere alla corrispettiva istanza di Cliente richiesta (N.B. la lista dei clienti è indicizzata sotto forma di mappa, dove la chiave d’accesso è la sua email: ciò permette, fra le altre cose, di velocizzare le operazioni di Login).

Inoltre, nel campo “carrello” è presente un ulteriore array: gli oggetti in esso contenuti presentano solo due campi numerici interi, “copia” e “componente”, i quali vengono usati per interrogare il Catalogo (pre-esistente) per trovare ciascuna istanza di Componente acquistata (cercandola per id) con la sua relativa CopiaComponente.

Infine, l’oggetto JSON dell’Ordine contiene tutti gli altri campi costanti richiesti per la sua costruzione: id, metodoPagamento, numeroCarta, cvv, città, indirizzo, CAP.

Nella sotto-cartella **“modello\_oggetti\_dominio”** è presente una copia *“pulita”* dei tre file, riempita con dei dati d’esempio ideali per permettere la dimostrazione del funzionamento del software.

## La classe Parser

Per costruire la classe Parser, si è deciso di utilizzare soli metodi e attributi *“static”*, in modo tale da renderla al pari di un servizio o una libreria di funzioni, anziché un effettivo oggetto software di cui andrebbe assegnata la responsabilità della gestione e creazione. In tale senso, non è stato previsto l’accesso esclusivo, e il Sistema e i suoi handler possono entrambi accedervi.

Per la scrittura dei metodi del Parser si è scelto di sfruttare la libreria esterna **org.json**.

Gli attributi al momento istanziati sono tre costanti, rappresentanti l’URL dei tre file citati al punto precedente. La classe è stata creata nel package **“data”**.

## Le funzioni di Caricamento

Una famiglia di funzioni dal nome nella forma generica *“caricaOggetto”* è stata creata con lo scopo di leggere i file del database all’avvio del sistema PCReady e caricarvi all’interno tutti gli oggetti necessari per il suo corretto funzionamento.

Fa eccezione la funzione per il caricamento del Catalogo, che per comodità di lettura è stata suddivisa in due funzioni: la funzione **“createCatalogo”**, chiamata dal sistema PCReady per ottenere il Catalogo dal database, che a sua volta chiama la funzione **“parseCatalogo”**, contenente una lunga lista di controlli – dovendo differenziare fra le diverse sotto-classi di Componente.

Le funzioni **“caricaClienti”** e **“caricaAdmin”** agiscono con la stessa logica e sullo stesso file (utenti.json) per creare in memoria, rispettivamente, le mappe dei Clienti e degli Amministratori.

La funzione **“caricaOrdini”** viene chiamata per ultima dal sistema, e richiede in ingresso la mappa degli Utenti e il Catalogo inizializzati dalle precedenti funzioni.

La logica interna di ciascuna di queste funzioni si basa su una famiglia di funzioni di parsing dal nome generico nella forma *“processOggetto”*, che – preso in ingresso un oggetto JSON – forniscono in uscita un’istanza di classe *Oggetto* corrispondente.

Nel caso specifico di **“processComponente”**, la funzione si limita a costruire un generico Componente a partire dal JSON, mentre la logica interna alla già citata *“parseCatalogo”* effettua uno switch-case sull’attributo “categoria” dell’oggetto: così facendo, può creare un’istanza nella sotto-classe specializzata, partendo dal Componente generico ottenuto da *“processComponente”* e aggiungendo i campi extra richiesti.

## Le funzioni di Salvataggio

In maniera perfettamente analoga, le funzioni di salvataggio hanno nome nella forma generica *“salvaOggetto”* (**salvaCatalogo**, **salvaUtenti**, **salvaOrdini**): prendendo le istanze degli oggetti tramite il sistema PCReady, si occupano di ri-generare i relativi file JSON partendo dai loro attributi, con una logica implementata attraverso una famiglia di funzioni apposite dal nome generico nella forma *“jsonOggetto”*.

Fa eccezione la funzione **“generalJson”**, che si occupa di fornire un JSON specializzato rispetto all’output della funzione **“jsonComponente”**, sulla base della sotto-classe specializzata del Componente che si vuole salvare su Database, completo degli attributi extra.

## Le funzioni di utility

La classe Parser contiene, infine, una famiglia di funzioni di “utility”, il cui scopo è solo quello di gestire la lettura e scrittura sui file che momentaneamente simulano il Database. In futuro, tali funzioni andrebbero sostituite con una famiglia corrispettiva che si occupi di gestire richieste HTTP, e pertanto sono state scritte in maniera tale da rendere quanto più banale e intuitiva suddetta futura transizione.

La funzione **“getFileContent”** si occupa semplicemente di ottenere il contenuto testuale di un file: suddetta funzione è stata specializzata nelle due funzioni **“getArrayFromFile”** e **“getObjectFromFile”**, che in aggiunta convertono automaticamente il contenuto del file, rispettivamente in un oggetto JSONArray o JSONObject, entrambi classi della libreria **org.json**.

La funzione **“writeToFile”** si occupa di sovrascrivere un file con una nuova stringa di testo, e quindi effettua l’operazione inversa di *“getFileContent”*. Per ottenere la rispettiva forma testuale di un JSONObject o di un JSONArray si utilizza il metodo standard **“toString”** prima di passarli come parametro. In questo caso specifico, *“toString”* accetta un parametro numerico intero, che indica il numero di spazi di cui è costituito un carattere *“tabulatore”*, nella formattazione del contenuto testuale: si è scelto ovunque di porre tale parametro pari a 4, nel rispetto della formattazione originale dei file presenti nella cartella *“modello\_oggetti\_dominio”*.

# **Interfaccia Utente: logica della Console**

Nell’intento di mostrare al cliente un software funzionante, e al contempo di creare una struttura che renda facile un lavoro futuro di espansione e di evoluzione del software, i nostri sforzi si sono concentrati in queste prime iterazioni nello sviluppo di un modello di interfaccia quanto più flessibile ed incrementale, al compromesso di implementare l’effettiva interfaccia – seppur momentaneamente – in sola forma testuale.

L’interfaccia è uguale per ciascun Utente, seppur specializzandosi in base al tipo di Utente che la utilizza (Cliente o Amministratore).

La nostra visione di interfaccia utente per questo software è la seguente:

* L’Utente dialoga con una **Console**, un terminale (testuale o grafico) che mette in comunicazione Utente e Sistema PC Ready.
* Più Utenti devono poter accedere contemporaneamente al Sistema, per cui più istanze di Console possono esistere contemporaneamente.
* La Console è specializzata in base all’Utente che la sta utilizzando.
* La Console è un “dispatcher di eventi”, ovvero un oggetto che – al pari di un robot – ha accesso a diverse azioni disponibili, che può chiamare in qualunque momento.
* Tali azioni vengono chiamate **Comandi**.
* Ciascun Comando ha accesso al Sistema PC Ready, è può quindi richiedergli di svolgere determinate funzioni su handler e oggetti su cui abbia una determinata responsabilità.
* Il Comando può essere “interno” (o nascosto), e svolgerà quindi sole funzioni di utility per altri Comandi, oppure può essere accessibile all’Utente.
* La lista di Comandi accessibili dall’Utente è gestita dalla classe **ElencoComandi**, che si occupa di riconoscere anche quali Comandi siano destinati ai soli Clienti, quali ai soli Amministratori, e quali ad entrambi.

Con questa visione siamo sicuri di poter mostrare al cliente un prodotto funzionante fin da subito, e che possa in futuro prendere qualunque specifica forma il cliente desideri, in base all’evoluzione dei suoi requisiti.

## La classe Comando

**Comando** è una classe “astratta” che implementa tre campi:

* Un **id** numerico, tale da poterlo distinguere univocamente dagli altri Comandi
* Una **descrizione** generica, da mostrare all’Utente tramite la Console, che spieghi in poche righe il compito da esso svolto
* Un campo booleano “adminOnly”, che – intuitivamente – permette alla Console di comprendere se un Utente di tipo Cliente abbia o meno diritto ad usufruire del Comando

Un Comando accessibile dall’Utente dovrà essere inizializzato fornendo tutti e tre i campi, mentre un Comando *“interno”* potrà essere inizializzato con dei valori di default.

Mentre tutte le sue funzioni pre-definite sono costituite dai getters/setters dei sopracitati campi, il suo *“cuore”* è la funzione astratta **“esegui”**, che prende come parametro un solo oggetto di tipo Console.

Ciascuno specifico Comando andrà definito creando una nuova classe a sé stante, che erediti dalla classe *“Comando”*, e che in quanto tale sia costretta a fornire una propria, esclusiva definizione della funzione *“esegui”*.

Si prenda – a titolo d’esempio – la classe “**ComandoSpegniConsole”**: tale Comando verrà chiamato quando l’Utente avrà intenzione di terminare le operazioni sul software, e verrà scelto dopo averne letto la descrizione *“Spegni la Console”*. Il corpo della sua funzione “esegui” si limita a fornire un messaggio d’addio all’Utente (“Grazie per aver usato PC Ready!”) e a chiamare le funzioni di spegnimento, sul Sistema – per garantire la permanenza dei nuovi dati – e sulla Console – per evitare che venga sfruttata da un Utente non autorizzato.

Inoltre, implementare ciascun Comando come una classe a sé stante anziché come una singola istanza di “Comando”, fornisce due vantaggi non indifferenti:

* Il programmatore ha la possibilità di definire le sue funzioni di utility da richiamare all’interno di “esegui”, per migliorare le operazioni di *testing*, *debugging* e *condivisione* e *lettura* del codice.
* L’aggiunta di un nuovo Comando equivale all’aggiunta di un nuovo file di classe nel progetto, richiedendo modifiche alla sola classe *ElencoComandi* e non a file critici quali le classi *“Console”*, *“PCReady”* o *“main”*.

## La classe ElencoComandi

La classe **ElencoComandi** si occupa di fornire alla classe Console un’interfaccia sugli specifici Comandi che andrebbero resi disponibili all’Utente, nascondendo al contempo l’esistenza dei Comandi “interni”.

Per far ciò, contiene due attributi *“static”* pari a due array di Comandi, una per i soli Clienti (**elencoCliente**) e una per i soli Amministratori (**elencoAmministratore**).

Le quattro funzioni sono – rispettivamente – due per ottenere uno specifico Comando dall’elenco, dato il suo id interno (**getComandoCliente** e **getComandoAmministratore**) e due per ottenere una forma testuale dell’elenco di Comandi disponibili (**stringCliente** e **stringAmministratore**).

## La classe Console

La classe Console è il fulcro principale dell’interattività fra Utente e Sistema, quella che implementa l’interfaccia utente a tutti gli effetti.

I suoi campi sono:

* **Sistema**: mantiene l’istanza singleton del Sistema PCReady per poterla sfruttare in qualunque momento.
* **On**: un booleano che determina se la Console è ancora utilizzabile, o se è stata “spenta” dall’Utente.
* **Admin**: un booleano che determina se l’Utente che ha accesso alla Console è un Amministratore o un regolare Cliente.
* **amministratoreCorrente**, **clienteCorrente**: due campi per tenere conto dell’istanza di Cliente/Amministratore a cui si sta fornendo l’accesso. Uno solo dei due campi viene inizializzato per ciascuna Console, alternativamente.

In aggiunta, è presente anche un campo di utility, lo **Scanner “in”**, poiché è richiesto per poter sfruttare le funzioni di input della libreria standard del linguaggio Java (per quelle di output è sufficiente accedere a System.out).

## Le funzioni di logica

La logica interna di ciascuna Console è gestita tramite due sole funzioni, *“accesso”* e *“esegui”*:

* **“accesso”** si occupa di svolgere, prima di qualunque altro, il compito del Comando **“ComandoLogin”**, ossia quello di autenticare l’Utente come Cliente o Amministratore, e in assenza di credenziali pre-esistenti di farlo registrare (tramite il Comando **“ComandoRegistrazione”**).  
    
  Una volta avvenuta una delle due operazioni con successo, la Console conoscerà il suo Cliente/Amministratore corrente, e potrà procedere all’esecuzione degli altri Comandi disponibili: quindi viene chiamata la funzione “esegui”.
* **“esegui”** si occupa di mostrare all’Utente la lista di Comandi a sua disposizione (grazie a quanto detto precedentemente per la classe ElencoComandi, qui utilizzata). Una volta immesso, da parte dell’Utente, un codice valido fra quelli a sua disposizione, verrà chiamata la funzione “esegui” della corrispondente istanza di Comando.  
    
  Tale operazione è svolta ciclicamente fintanto che l’utente non chiama il Comando *“ComandoSpegniConsole”*.

## Le funzioni di utility

La Console contiene al suo interno una famiglia di funzioni di Utility che i vari Comandi possono sfruttare per interagire con l’Utente. Poiché l’implementazione attuale della Console è sotto la forma di interfaccia testuale, tali funzioni si limitano ad effettuare un “wrapper” sopra le funzioni classiche di *input* (classe Scanner) e di *output* (System.out) fornite dal linguaggio.

La funzione **“print”** si limita – banalmente – a mostrare a schermo un messaggio testuale.

La famiglia di funzioni dal nome generico *“getTipo”* si occupa di valutare l’input testuale dell’Utente come una variabile dello specifico Tipo scelto, ed eventualmente di richiedere un nuovo input qualora quello analizzato non sia valido, mostrando uno specifico messaggio d’errore.

Ciascuna di queste funzioni presenta una variante (**override**) che accetta un parametro testuale in ingresso: tale stringa viene utilizzata come richiesta da mostrare a schermo all’utente per richiedere l’input, anziché stampare una richiesta generica (i.e. *“Inserire un numero intero”* nel caso della funzione **“getInt”**, che con l’override può essere sostituito da un messaggio custom).

Di queste funzioni, l’unica di particolare interesse è **“getYesNo”**, che trasporta il linguaggio informatico del tipo Booleano (true / false) nel linguaggio umano delle domande a risposta Sì/No, e che quindi implementa una logica leggermente più specifica per valutare l’input umano dell’Utente.

Le rimanenti funzioni della classe Console sono composte da getters e setters auto-generati.