**Università degli Studi di Salerno**
**Corso di Ingegneria del Software**

# ClickFly
# Object Design Document
# Versione 1.4

Data: 21/01/2026

**Coordinatore del Progetto:**

| Nome | Matricola |
|---|---|
| Fabio Pennarella | |

**Partecipanti:**

| Nome | Matricola |
|---|---|
| Fabio Pennarella | |
| Giusy Chierchia | 0512117508 |

# Revision History

| Data | Versione |
|---|---|
| 02/01/2026 | 1.0 |
| 03/01/2026 | 1.1 |
| 04/01/2026 | 1.2 |
| 05/01/2026 | 1.3 |

| | |
|---|---|
| 24/01/2026 | 1.4 |

# Introduzione

## 1.1 Object Design Trade-offs

Il design di **ClickFly** si basa sul pattern architetturale **MVC (Model-View-Controller)**, adottato per garantire una chiara separazione delle responsabilità tra presentazione, logica applicativa e gestione dei dati.

- **Buy vs Build**:
  È stato adottato un approccio *Buy* per componenti non critiche del sistema, come **Bootstrap** per il front-end e **Apache Tomcat** come application server, al fine di ridurre i tempi di sviluppo e aumentare l'affidabilità complessiva della piattaforma.
  Le funzionalità core del sistema, come **ricerca voli, gestione del carrello, prenotazioni e pagamenti**, sono state invece sviluppate internamente (*Build*) per garantire pieno controllo sulla logica applicativa.

- **Memory Space vs Response Time**:
  Il sistema privilegia tempi di risposta rapidi attraverso l'uso di strutture dati in memoria, come la **sessione utente** per la gestione del carrello e dello stato di autenticazione.
  Questa scelta comporta un maggiore utilizzo della memoria, accettato per ridurre l'accesso frequente al database e migliorare le prestazioni complessive del sistema.

---

## 1.2 Linee guida per la documentazione dell'interfaccia

Le seguenti convenzioni sono state adottate per uniformare il design delle interfacce e garantire coerenza e manutenibilità del codice.

### Naming

- Classi con nomi al singolare e descrittivi (es. `Volo`, `Prenotazione`, `Utente`).

- Metodi con frasi verbali che indicano chiaramente l'azione svolta (es. `aggiungiVolo()`, `confermaPrenotazione()`, `validaPagamento()`).

- Campi e parametri con nomi sostantivi esplicativi (es. `prezzo`, `idVolo`, `postiDisponibili`, `emailUtente`).

### Gestione degli errori

- Gli errori sono segnalati tramite **eccezioni**, opportunamente gestite a livello di Service Layer.

- Le operazioni su collezioni restituiscono oggetti di tipo **List** o **Map**, garantendo robustezza rispetto a modifiche e iterazioni sugli elementi.

### Conformità

- Tutte le interfacce pubbliche devono essere documentate tramite **JavaDoc**.

- Le interfacce seguono il principio di **interfacce minime**, esponendo solo i metodi strettamente necessari.

---

## 1.3 Definizioni, acronimi e abbreviazioni

- **DAO (Data Access Object)**: pattern utilizzato per separare la logica di accesso ai dati dal resto dell'applicazione.

- **DTO (Data Transfer Object)**: oggetto utilizzato per il trasferimento dei dati tra i diversi layer del sistema.

- **MVC (Model-View-Controller)**: pattern architetturale che separa la rappresentazione dei dati, la logica applicativa e la gestione dell'interazione con l'utente.

---

## 1.4 Riferimenti

- Requirements Analysis Document (RAD)

- System Design Document (SDD)

- Problem Statement Document

---

# 2. Packages

## 2.1 Struttura dei pacchetti

Il sistema **ClickFly** è suddiviso nei seguenti pacchetti principali:

### 1. Presentation Layer

- **Pacchetto**: `com.clickfly.presentazione`
- **Componenti principali**:
  `VistaHome`, `VistaRicercaVoli`, `VistaCarrello`, `VistaPrenotazioni`, `VistaLogin`

### 2. Service Layer

- **Pacchetto**: `com.clickfly.servizio`
- **Componenti principali**:
  `ServizioAutenticazione`, `ServizioVoli`, `ServizioPrenotazioni`, `ServizioPagamenti`

### 3. Persistence Layer

- **Pacchetto**: `com.clickfly.persistenza`
- **Componenti principali**:
  `UtenteDAO`, `VoloDAO`, `PrenotazioneDAO`, `PagamentoDAO`

Ogni pacchetto è strutturato in modo da **ridurre le dipendenze** tra i componenti e **favorire il riuso del codice**, in linea con i principi di modularità e manutenibilità.

---

## 2.2 Struttura grafica dei packages nel sistema

In questa sezione è descritta la struttura organizzativa dei file all'interno del progetto **ClickFly**.
Il progetto è organizzato come una **Java Web Application** che utilizza **Maven** come gestore delle dipendenze.

La suddivisione dei sottosistemi individuata nel **System Design Document** viene pienamente rispettata e implementata tramite i package:

- `presentazione` (Presentation Layer)
- `servizio` (Service Layer)
- `persistenza` (Persistence Layer)

All'interno di ciascun package sono realizzati i componenti definiti nel **component diagram**, rispettando le responsabilità assegnate a ogni layer.

Il progetto include inoltre la struttura standard Maven, come:

- directory `target`
- file `pom.xml`
- file di configurazione

Il tutto rispecchia l'organizzazione tipica di un progetto che utilizza le seguenti tecnologie:
**Java Servlet, JSP, Bootstrap, Maven, HTML, CSS**.

# Struttura dei package (ClickFly)

Il progetto ClickFly è organizzato secondo una classica architettura **MVC a 3 layer** (Presentation / Service / Persistence), in modo da separare chiaramente responsabilità e dipendenze.

### Package `com.clickfly.modello`

Contiene le **Java Beans (entità di dominio)** che rappresentano gli oggetti definiti nel class diagram e nell'ER diagram (es. `Utente`, `Volo`, `Carrello`, `Prenotazione`, ecc.).
Queste classi incapsulano attributi e metodi base (getter/setter) e vengono usate da tutti gli altri layer.

### Package `com.clickfly.presentazione`

Contiene la **parte di controllo (Controller)** del pattern MVC, quindi:

- **Servlet** che ricevono richieste HTTP dal client (pagine JSP o form),
- validano input (o delegano al Service Layer),
- inoltrano la richiesta verso la vista corretta (forward/redirect),
- gestiscono sessione (`sessionId`) per utente loggato e contenuti del carrello.

Esempi tipici: servlet per **login, registrazione, ricerca voli, gestione carrello, checkout/prenotazione, "le mie prenotazioni"**.

## Package `com.clickfly.servizio`

Contiene la **logica di business** del sistema (Service Layer).
Qui vengono implementate le regole applicative principali, ad esempio:

- autenticazione e gestione sessione utente,
- aggiunta/rimozione voli dal carrello,
- creazione prenotazione e calcolo totale,
- controllo disponibilità posti,
- gestione portafoglio/ricarica (se prevista),
- validazioni "di processo" (non solo di form).

Le servlet **non devono parlare direttamente col database**, ma chiamano i service.

## Package `com.clickfly.persistenza`

Contiene l'accesso ai dati (Persistence Layer), quindi:

- **DAO Interface + DAO Implementation**
- gestione connessione database (MySQL)
- query e operazioni CRUD su tabelle: utenti, voli, carrello, prenotazioni, ecc.

Le classi DAO forniscono metodi usati dal Service Layer per leggere/scrivere informazioni persistenti.

---

## Struttura `webapp`

Nel modulo `webapp` sono presenti tutte le risorse web del progetto:

- cartella **`jsp/`** con le pagine JSP (Home, Cerca voli, Login, Registrazione, Carrello, Le mie prenotazioni…)
- **`META-INF/`** e **`WEB-INF/`** per configurazioni e deployment (es. `web.xml`, eventuali configurazioni del container)
- cartella **`resources/`** con:
    - `images/` (logo ClickFly, icone…)
    - `styles/` (CSS, Bootstrap/custom)
    - `scripts/` (JS)
    - `templates/` (componenti riutilizzabili come **header e footer**, usati in quasi tutte le pagine)

Questa organizzazione riflette la separazione dei sottosistemi descritta nel System Design e rende il progetto più mantenibile e scalabile.

---

## (Aggancio alla sezione successiva)

**3. Interfacce delle classi**
Per ogni package verranno fornite in questa sezione le principali interfacce pubbliche e i metodi più rilevanti (Servlet → Service → DAO), descrivendo input, output ed eccezioni gestite per le funzionalità di ClickFly (registrazione, login, ricerca voli, carrello, prenotazione, gestione prenotazioni).

## Interfacce delle classi

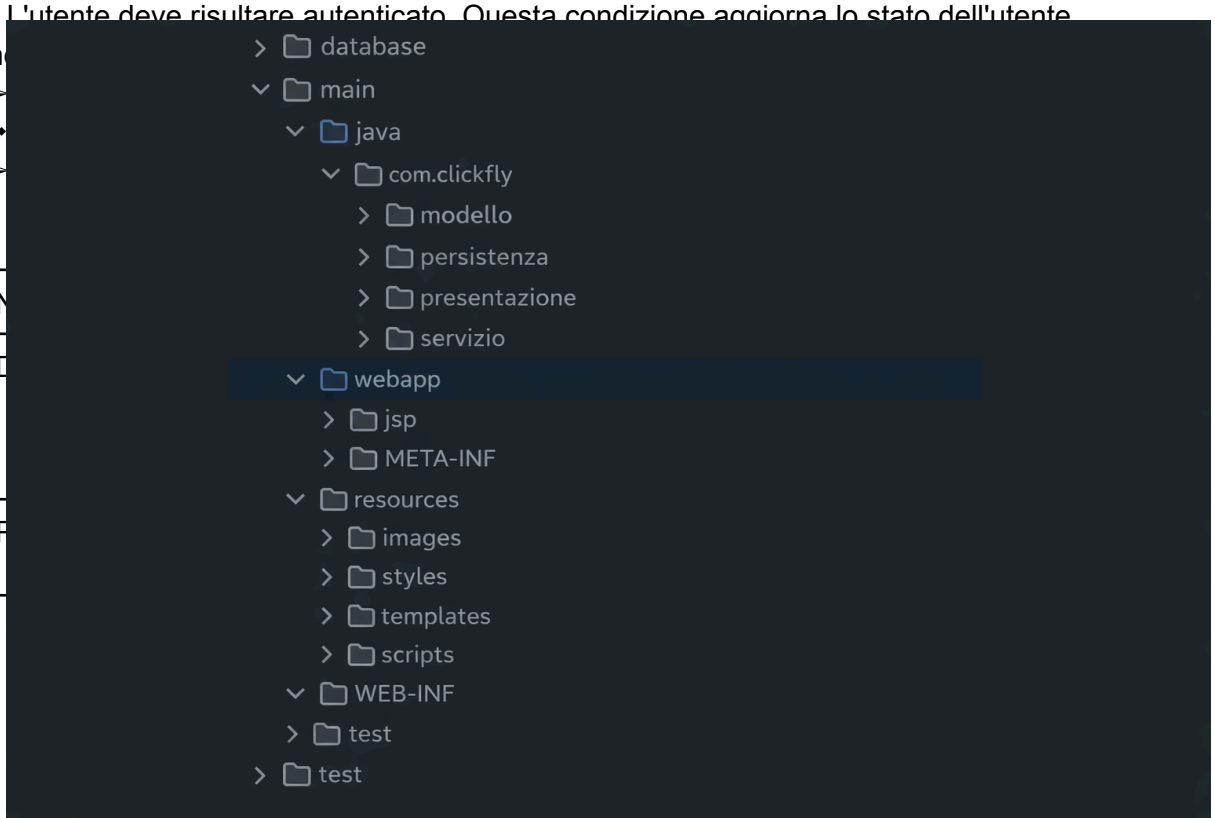
Per ogni package verrà fornito in questa sezione le interfacce pubbliche e i relativi metodi principali.

| Interfaccia | Authentication Service |
|---|---|
| Descrizione | Authentication Service fornisce il servizio di autenticazione degli utenti,verifica dei ruoli controllo delle credenziali |
| Metodi | +login(Stringemail,Stringpassword):user<br>+logout(Stringemail):void<br>+getRuolo(Stringemail):String |
| Invariante di classe | -Gli utenti non autenticati non devono avere ruoli accessibili - Consistenza del ruolo |

| Nome Metodo | +login(Stringemail,Stringpassword):user |
|---|---|
| Descrizione | Questo metodo permette di effettuare il login, controllandole credenziali e creandola sessione per l'utente |
| Pre Condizioni | contextAuthenticationService::login(email:String,password:String):User<br>pre:<br>notemail.isEmpty()andnotpassword.isEmpty()<br>andself.isEmailValid(email) |
| Post Condizioni | contextAuthenticationService::login(email:String,password:String):User<br>post: |
| | result<>null implies<br>result.email=emailand<br>self.verifyPassword(password,result.getHashedPassword())=true<br>and<br>result.isAuthenticated=true<br>and<br>self.sessions->includes(result) |

| | |
|---|---|
| Invarianti | contextUser<br>inv:self.hashedPassword->notEmpty()<br>andself.hashedPassword=hash(self.hashedPassword) |

Spiegazioni su:
❖Pre-condizioni:
➢ L'email e la password non devono essere vuote.
➢ L'email deve essere valida secondo un criterio definito nel metodo isEmailValid.
❖Post-condizioni:
➢Se il risultato(result)non è null, significa che il login è avvenuto con successo.
Inoltre:
■ L'utente restituito dal metodo deve avere l'email corrispondente a quella fornita in ingresso.
■ L'utente deve risultare autenticato. Questa condizione aggiorna lo stato dell'utente in



| | |
|---|---|
| | **AuthenticationService::logout(email: String, password: String): void**<br><br>**pre:**<br><br>● `email` non è vuota<br>● `password` non è vuota<br>● Esiste una sessione `s` tale che:<br>○ `s.user.email = email`<br>○ `s.user.isAuthenticated = true`<br>● La password fornita è corretta:<br>○ `verifyPassword(password, getUserByEmail(email).hashedPassword) = true` |
| Post Condizioni | **context AuthenticationService::logout(email: String, password: String): void**<br><br>**post:**<br><br>● Non esiste alcuna sessione associata all'utente con email = `email`<br>○ `sessions->forAll(s \| s.user.email <> email)` |
| Invarianti | ## Invarianti<br><br>**context AuthenticationService**<br><br>**inv:**<br><br>● Ogni utente autenticato deve avere una sessione attiva: |

| | |
|---|---|
| | ○ `users->select(u \| u.isAuthenticated = true) ->forAll(u \| sessions->exists(s \| s.user = u))` |

**Pre-condizioni**

- L'email e la password fornite dall'utente non devono essere vuote.
- Deve esistere una sessione attiva associata all'utente identificato dall'email inserita.
- L'utente deve risultare autenticato nel sistema (`isAuthenticated = true`).
- La password fornita deve corrispondere all'hash memorizzato nel sistema per l'utente, al fine di garantire la validità dell'operazione di logout.

---

❖ **Post-condizioni**

- Al termine dell'esecuzione del metodo `logout`, non devono essere presenti sessioni attive associate all'utente identificato dall'email fornita.
- L'utente viene correttamente disconnesso dal sistema ClickFly.

---

❖ **Invarianti**

- Per ogni utente che risulta autenticato nel sistema (`isAuthenticated = true`), deve esistere una e una sola sessione attiva associata a tale utente.
- Il sistema non deve consentire la presenza di utenti autenticati privi di una sessione valida.

| | |
|---|---|
| Nome Metodo | +getRuolo(Stringemail):String |

| | |
|---|---|
| Descrizione | Questo metodo permette di ottenere il ruolo e riconoscere che ruolo l'utente che è autenticato |
| Pre Condizioni | L’email e la password fornite dall’utente non devono essere vuote.<br><br>Deve esistere una sessione attiva associata all’utente identificato dall’email inserita.<br><br>L’utente deve risultare autenticato nel sistema (`isAuthenticated = true`).<br><br>La password fornita deve corrispondere all’hash memorizzato nel sistema per l’utente, al fine di garantire la validità dell’operazione di logout. |
| Post Condizioni | • Al termine dell’esecuzione del metodo `logout`, non devono essere presenti sessioni attive associate all’utente identificato dall’email fornita.<br><br>• L’utente viene correttamente disconnesso dal sistema ClickFly. |
| Invarianti | • Per ogni utente che risulta autenticato nel sistema (`isAuthenticated = true`), deve esistere una e una sola sessione attiva associata a tale utente.<br><br>• Il sistema non deve consentire la presenza di utenti autenticati privi di una sessione valida. |

**Pre-condizioni**

- L’email fornita non deve essere vuota.
- Deve esistere una sessione attiva associata all’utente identificato dall’email inserita.
- L’utente deve risultare autenticato nel sistema ClickFly.

---

**Post-condizioni**

- Il valore restituito (`result`) deve corrispondere al ruolo associato all’utente identificato dall’email fornita.
- Il ruolo restituito deve essere coerente con i permessi assegnati all’utente all’interno del sistema ClickFly.

---

**Invarianti**

- Ogni utente registrato nel sistema ClickFly deve avere esattamente un ruolo assegnato.
- Il ruolo associato a un utente non deve mai essere nullo né vuoto durante il ciclo di vita dell’utente nel sistema.

| Interfaccia | `SecurityService` |
|---|---|
| Descrizione | `SecurityService` fornisce i servizi per la protezione dei dati sensibili nel sistema ClickFly, in particolare per la gestione sicura delle credenziali degli utenti.<br>Il servizio implementa meccanismi di hashing delle password e verifica delle credenziali, garantendo che nessuna password venga mai memorizzata o trasmessa in chiaro. |
| Metodi | **`hashPassword(String password) : String`**<br>Restituisce l’hash sicuro della password fornita utilizzando un algoritmo di hashing robusto (es. bcrypt). |

| | |
|---|---|
| | **`verifyPassword(String insertedPassword, String hashedPassword) : boolean`**<br>Verifica che la password inserita dall'utente corrisponda all'hash memorizzato nel sistema.<br><br>**`getHashedPassword(String email) : String`**<br>Restituisce la password hashata associata all'utente identificato dall'email fornita. |
| Invariante di classe | Ogni password nel sistema ClickFly deve essere memorizzata esclusivamente in forma hashata.<br><br>L'hash restituito non deve mai essere nullo né vuoto.<br><br>Nessuna operazione del sistema deve richiedere l'accesso alla password in chiaro. |

Elenco,descrizione,spiegazioni metodi

| Nome Metodo | + hashPassword(String password) : String |
|---|---|
| Descrizione | Questo metodo consente di eseguire l'hashing della password inserita dall'utente all'interno del sistema **ClickFly**, utilizzando un algoritmo di hashing sicuro.La password non viene mai salvata né restituita in chiaro, garantendo la protezione delle credenziali dell'utente. |
| Pre Condizioni | **context**<br>**`SecurityService::hashPassword(password : String) : String`**<br><br>**pre:**<br><br>● La password fornita non deve essere vuota<br><br>`not password.isEmpty()` |

| | |
|---|---|
| Post Condizioni | **context** `SecurityService::hashPassword(password : String) : String`<br><br>**post:**<br><br>● Il valore restituito non deve essere vuoto<br><br>● Il valore restituito deve corrispondere all'hash della password fornita<br><br>`result <> ""`<br>`and result = hash(password)` |
| Invarianti | **context** `SecurityService`<br><br>**inv:**<br><br>● Il valore hash generato non deve mai essere vuoto<br><br>● L'hash deve essere sempre coerente con la funzione di hashing applicata alla password<br><br>`not result.isEmpty()`<br>`and result = hash(password)` |

## Pre-condizioni

- La password fornita dall'utente **non deve essere vuota**, in quanto l'algoritmo di hashing di ClickFly richiede un input valido per generare una stringa hash corretta.

---

## Post-condizioni

- Il risultato restituito dal metodo è **una stringa non vuota**.
- Il valore restituito rappresenta **la versione hashata della password**, generata tramite l'algoritmo di hashing sicuro adottato dal sistema.

---

## Invarianti

- L'output del metodo `hashPassword` **non può mai essere vuoto**: l'invariante garantisce che il metodo restituisca sempre una stringa valida.
- La funzione di hashing deve essere **deterministica rispetto all'input**, ovvero, a parità di password fornita, la funzione produce sempre un hash coerente secondo l'algoritmo utilizzato dal sistema ClickFly.

| | |
|---|---|
| Nome Metodo | `+ verifyPassword(String insertedPassword, String hashedPassword) : boolean` |
| Descrizione | Questo metodo verifica se la password inserita dall'utente durante la fase di autenticazione corrisponde alla password hashata memorizzata nel sistema ClickFly. La verifica avviene applicando l'algoritmo di hashing alla password inserita e confrontando il risultato con l'hash salvato. |
| Pre Condizione | **context SecurityService::verifyPassword(insertedPassword : String, hashedPassword : String) : boolean**<br>`pre:`<br>`  not insertedPassword.isEmpty() and`<br>`  not hashedPassword.isEmpty()` |

| | |
|---|---|
| Post Condizione | **context SecurityService::verifyPassword(insertedPassword : String, hashedPassword : String) : boolean**<br><br>`post:`<br>`  result = ( hash(insertedPassword) = hashedPassword )` |
| Invariante | **context SecurityService::verifyPassword(insertedPassword : String, hashedPassword : String) : boolean**<br><br>`inv:`<br>`  not insertedPassword.isEmpty() and`<br>`  not hashedPassword.isEmpty() and`<br>`  result = ( hash(insertedPassword) = hashedPassword )` |

## Spiegazioni

❖ **Pre-condizioni:**

➢ Assicurano che entrambe le password, quella inserita dall'utente e quella hashata memorizzata nel sistema ClickFly, non siano vuote.

---

❖ **Post-condizioni:**

➢ Verificano che l'hash della password inserita corrisponda alla password hashata memorizzata nel sistema, restituendo `true` in caso di corrispondenza e `false` in caso contrario.

---

❖ **Invarianti:**

➢ L’invariante garantisce che entrambe le password non siano vuote, poiché non è significativo effettuare una verifica su valori nulli o vuoti.

➢ Il risultato della verifica deve essere sempre coerente con l’operazione di hashing adottata dal sistema ClickFly: il valore `result` è `true` se e solo se l’hash della password inserita coincide con quello memorizzato nel sistema.

| Nome Metodo | + getHashedPassword(String email) : String |
|---|---|
| Descrizione | Questo metodo restituisce la password hashata associata a un utente del sistema **ClickFly**, identificato in modo univoco tramite il proprio indirizzo email. |
| Pre Condizioni | Questo metodo restituisce la password hashata associata a un utente del sistema **ClickFly**, identificato in modo univoco tramite il proprio indirizzo email. |
| Post Condizioni | **context SecurityService::getHashedPassword(email : String) : String**<br>**post:**<br>• `result = self.users->select(u \| u.email = email).hashedPassword` |
| Invarianti | **context SecurityService::getHashedPassword(email : String) : String**<br>**inv:**<br>• `not email.isEmpty()`<br>• `self.users->exists(u \| u.email = email)` |

| | |
|---|---|
| | • `not result.isEmpty()`<br>• `result = self.users->select(u \| u.email = email).hashedPassword` |

## Spiegazioni

❖ **Pre-condizioni:**

➢ Assicura che l'email fornita non sia vuota.
➢ Assicura che esista un utente registrato nel sistema ClickFly associato all'email specificata.

❖ **Post-condizioni:**

➢ Restituisce la password hashata dell'utente corrispondente all'email fornita.

❖ **Invarianti:**

➢ L'email utilizzata per l'operazione deve sempre essere valida e riferita a un utente esistente nel sistema.
➢ La password restituita deve essere sempre non vuota.
➢ Il valore restituito deve corrispondere alla password hashata memorizzata per l'utente nel sistema ClickFly.

| **Interfaccia** | **CartService** |
|---|---|
| Descrizione | CartService fornisce i servizi per la gestione del **carrello voli** di ClickFly.<br>Permette all'utente di aggiungere voli al carrello, rimuoverli o modificarne la quantità di posti selezionati prima di procedere alla prenotazione. |
| Metodi | **+ aggiungiItem(Volo v, int postiSelezionati) : boolean** |

| | |
|---|---|
| | **+ rimuoviItem(Volo v) : void**<br><br>**+ modificaItem(Volo v, int postiSelezionati) : boolean** |
| Invariante di classe | • Ogni volo presente nel carrello deve essere un volo valido e disponibile nel sistema.<br><br>• Ogni volo nel carrello deve avere un numero di posti selezionati strettamente positivo.<br><br>• Ogni volo può comparire **una sola volta** nel carrello dell'utente.<br><br>• Il numero di posti selezionati per un volo non deve superare i posti disponibili per quel volo. |

| | |
|---|---|
| **Nome Metodo** | + aggiungiItem(Volo v, int postiSelezionati) : boolean |
| **Descrizione** | Questo metodo permette di aggiungere un **volo** al carrello dell'utente.<br>Se il volo è già presente nel carrello, il metodo aggiorna il numero di posti selezionati; in caso contrario, il volo viene aggiunto come nuovo elemento del carrello, nel rispetto della disponibilità dei posti. |
| **Pre Condizione** | **context CartService::aggiungiItem(v:Volo, postiSelezionati:int) : boolean**<br><br>`pre:`<br>`v <> null`<br>`and postiSelezionati > 0`<br>`and v.postiDisponibili >= postiSelezionati`<br>`and v.isDisponibile = true` |

| | |
|---|---|
| **Post Condizione** | **context CartService::aggiungiItem(v:Volo, postiSelezionati:int) : boolean**<br><br>`post:`<br>`if self.carrello->exists(item \| item.volo = v) then`<br>`  item.postiSelezionati = item.postiSelezionati + postiSelezionati`<br>`  and item.postiSelezionati <= v.postiDisponibili`<br>`  and result = true`<br>`else`<br>`  self.carrello->includes(`<br>`  CartItem { volo = v, postiSelezionati = postiSelezionati }`<br>`  )`<br>`  and result = true`<br>`endif` |
| **Invarianti** | // |

Spiegazioni su:
Pre-condizioni:  Il prodotto non deve essere null e deve essere valido (disponibile nel sistema).
Post-condizioni: Se il prodotto già presente nel carrello, l'aggiunta deve riuscire, altrimenti l'operazione restituisce false

| | |
|---|---|
| **Nome Metodo** | + rimuoviItem(Volo v) : void |
| **Descrizione** | Questo metodo permette di rimuovere un **volo** dal carrello dell'utente.<br>Se il volo non è presente nel carrello, non viene effettuata alcuna modifica. |
| **Pre Condizioni** | **context CartService::rimuoviItem(v:Volo) : void**<br><br>`pre:` |

| | |
|---|---|
| | `v <> null`<br>`and self.carrello->exists(item \|`<br>`item.volo = v)` |
| **Post Condizioni** | **context CartService::rimuoviItem(v:Volo) : void**<br><br>`post:`<br>`not self.carrello->exists(item \|`<br>`item.volo = v)` |
| **Invarianti** | |

Spiegazioni su:

Pre-condizioni:
Il prodotto deve essere presente nel carrello e non può essere null.
Post-condizioni:
Una volta che il prodotto è stato rimosso,non deve più essere presente nel carrello

| | |
|---|---|
| **Nome Metodo** | + modificaItem(Volo v, int posti) : boolean |
| **Descrizione** | Questo metodo permette di modificare il **numero di posti selezionati** per un volo presente nel carrello.<br>Se il numero di posti è valido (positivo e compatibile con la disponibilità del volo), la modifica viene applicata; in caso contrario, l'operazione non viene eseguita. |
| **Pre Condizione** | **context CartService::modificaItem(v:Volo, posti: Integer) : boolean**<br><br>`pre:`<br>`v <> null`<br>`and self.carrello->exists(item \|`<br>`item.volo = v)`<br>`and posti > 0`<br>`and v.postiDisponibili >= posti` |
| **Post Condizioni** | **context** |

| | |
|---|---|
| | **CartService::modificaItem(v:Volo, posti: Integer) : boolean**<br><br>`post:`<br>`if self.carrello->exists(item \| item.volo = v)`<br>`then`<br>`  item.postiSelezionati = posti`<br>`  and result = true`<br>`else`<br>`  result = false` |
| Invarianti | // |

Spiegazioni su:
Pre-condizioni: Il prodotto non deve essere nullo,deve essere presente nel carrello e la quantità deve essere positiva.
Post-condizioni: Se la modifica della quantità è riuscita, il metodo restituisce true,altrimenti false.

| Interfaccia | OrderService |
|---|---|
| Descrizione | **OrderService** fornisce i servizi per la gestione delle **prenotazioni di voli**, includendo la **creazione di una prenotazione** a partire dal carrello e la **gestione dello stato della prenotazione** durante il suo ciclo di vita. |
| Metodi | + createOrder(Collection<Volo> voli, Utente u, Date dataPrenotazione, double totale) : Prenotazione<br>+ modifyStatus(Prenotazione p, String nuovoStato) : boolean<br>+ getStatus(Prenotazione p) : String |
| Invariante di classe | - Le prenotazioni devono avere stati validi (es. IN_ELABORAZIONE, CONFERMATA, ANNULLATA)<br>- Le prenotazioni devono avere un totale positivo |

| | |
|---|---|
| | - Le prenotazioni devono essere associate a utenti validi e autenticati<br>- Ogni prenotazione deve contenere almeno un volo |

## Pre-condizioni:

- La collezione di **voli** non può essere **null** né **vuota**.
- L'**utente** associato alla prenotazione deve essere **valido e autenticato** nel sistema.
- La **data della prenotazione** non può essere precedente alla data corrente.
- Il **totale** della prenotazione deve essere **maggiore di zero**.

---

## ❖ Post-condizioni:

- Viene creata una **nuova prenotazione** contenente i dettagli forniti in ingresso.
- La prenotazione è **associata all'utente** e include **tutti i voli specificati**.
- Lo **stato iniziale** della prenotazione viene impostato correttamente (es. *IN_ELABORAZIONE*).

| **Nome Metodo** | +modifyStatus(Prenotazione p) : boolean |
|---|---|
| **Descrizione** | Questo metodo consente di **modificare lo stato di una prenotazione** nel sistema ClickFly (ad esempio da *IN_ELABORAZIONE* a *CONFERMATA*, oppure a *CANCELLATA*), nel rispetto delle regole di transizione definite. |
| **Pre Condizioni** | **context OrderService::modifyStatus(p : Prenotazione) : boolean**<br><br>`pre:`<br>`p <> null` |

| | |
|---|---|
| | `and`<br>`self.prenotazioni->includes(p)`<br>`and p.stato.isModificabile()` |
| **Post Condizioni** | **context OrderService::modifyStatus(p : Prenotazione) : boolean**<br><br>`post:`<br>`if result = true then`<br>`  p.stato = p.stato.next()`<br>`endif` |
| **Invarianti** | // |

Spiegazioni su:
Pre-condizioni:

- L'ordine non deve essere null.
- L'ordine deve esistere nel sistema.
- Lo stato attuale dell'ordine deve permettere modifiche.

Post-condizioni:

- Lo stato dell'ordine viene modificato,se possibile.
- Restituisce true se la modifica è stata eseguita, false altrimenti.

| Nome Metodo | +getStatus(Prenotazione p) : String |
|---|---|
| Descrizione | Questo metodo restituisce **lo stato attuale di una prenotazione** nel sistema ClickFly (ad esempio *IN_ELABORAZIONE*, *CONFERMATA*, *CANCELLATA*, *COMPLETATA*). |
| Pre-Condizione | **context OrderService::getStatus(p : Prenotazione) : String**<br><br>`pre:`<br>`p <> null`<br>`and`<br>`self.prenotazioni->includes(p)` |
| Post-Condizione | **context OrderService::getStatus(p :** |

| | |
|---|---|
| | **Prenotazione) : String**<br><br>`post:`<br>`result = p.stato` |
| Invariante | // |

Spiegazioni su:

Pre-condizioni:
- L'ordine non deve essere null.
- L'ordine deve esistere nel sistema.

Post-condizioni:
- Restituisce lo stato dell' ordine specificato.

| **Interfaccia** | **Catalog Service** |
|---|---|
| Descrizione | CatalogService fornisce i servizi per la **gestione del catalogo dei voli** disponibili su ClickFly.<br>Consente agli operatori di sistema di **creare, modificare e rimuovere voli**, garantendo la coerenza dei dati e la corretta disponibilità dei posti. |
| Metodi | + creaVolo(<br>idVolo : String,<br>partenza : String,<br>destinazione : String,<br>data : Date,<br>orario : Time,<br>prezzo : Double,<br>compagniaAerea : String,<br>postiDisponibili : int<br>) : Volo<br><br>+ modificaVolo(v : Volo) : boolean<br><br>+ rimuoviVolo(v : Volo) : boolean |
| Invariante di classe | // |

| | |
|---|---|
| Nome Metodo | + creaVolo(<br>String idVolo,<br>String aeroportoPartenza,<br>String aeroportoArrivo,<br>Date dataPartenza,<br>Date dataArrivo,<br>double prezzo,<br>String compagniaAerea,<br>int postiDisponibili<br>) : Volo |
| Descrizione | Questo metodo crea un nuovo volo nel catalogo di ClickFly utilizzando i dettagli forniti, rendendolo disponibile per la ricerca e la prenotazione. |
| Pre Condizione | context CatalogService::creaVolo(<br>idVolo : String,<br>aeroportoPartenza : String,<br>aeroportoArrivo : String,<br>dataPartenza : Date,<br>dataArrivo : Date,<br>prezzo : Double,<br>compagniaAerea : String,<br>postiDisponibili : Integer<br>) : Volo<br><br>pre:<br>not idVolo.isEmpty()<br>and not aeroportoPartenza.isEmpty()<br>and not aeroportoArrivo.isEmpty()<br>and not compagniaAerea.isEmpty()<br>and prezzo > 0<br>and postiDisponibili >= 0<br>and dataPartenza >= Date::now() |
| Post Condizione | context CatalogService::creaVolo(<br>idVolo : String,<br>aeroportoPartenza : String,<br>aeroportoArrivo : String,<br>dataPartenza : Date,<br>dataArrivo : Date,<br>prezzo : Double, |

| | |
|---|---|
| | compagniaAerea : String,<br>postiDisponibili : Integer<br>) : Volo<br><br>post:<br>result <> null<br>and self.catalogo->includes(result) |
| Invarianti | context CatalogService<br>inv:<br>self.catalogo->forAll(v \|<br>not v.idVolo.isEmpty()<br>and v.prezzo > 0<br>and v.postiDisponibili >= 0<br>) |

Spiegazionisu:
Pre-condizioni:
Il prezzo deve essere maggiore di zero.
Gli identificativi di prodotto(id,description,brand, figure)non possono essere nulli vuoti.
❖Post-condizioni: ➢ Il prodotto creato è presente nel catalogo.

| | |
|---|---|
| **Nome Modello** | + modificaVolo(Volo v) : boolean |
| **Descrizione** | Questo metodo consente di modificare i dettagli di un volo esistente nel catalogo di ClickFly, come orari, prezzo o disponibilità dei posti. |
| **Pre Condizioni** | context CatalogService::modificaVolo(v : Volo) : boolean<br>pre:<br>v <> null<br>and self.catalogo->includes(v) |
| **Post Condizioni** | context CatalogService::modificaVolo(v : Volo) : boolean<br>post: |

| | |
|---|---|
| | if result = true then<br>self.catalogo->includes(v)<br>endif |
| **Invarianti** | // |

## Pre-condizioni:

➢ Il **volo** non deve essere `null`.
➢ Il **volo** deve esistere nel catalogo dei voli del sistema ClickFly.

## ❖ Post-condizioni:

➢ Il **volo** viene aggiornato con i nuovi dettagli forniti (es. orari, prezzo, disponibilità posti).
➢ Il metodo restituisce **true** se l'operazione di modifica va a buon fine, **false** altrimenti.

| Nome Metodo | +rimuoviVolo(Volo v): boolean |
|---|---|
| Descrizione | Questo metodo rimuove un **volo esistente** dal catalogo dei voli del sistema ClickFly. L'operazione consente agli operatori di gestione di eliminare voli non più disponibili o non più validi. |
| Pre Condizione | context CatalogService::rimuoviVolo(v: Volo): boolean<br>pre:<br>v <> null<br>and self.catalogoVoli->includes(v)<br>➢ Il volo non deve essere null.<br>➢ Il volo deve esistere nel catalogo dei voli del sistema. |
| Post Condizione | Il catalogo dei voli non può contenere voli duplicati.<br>➢ Ogni volo presente nel catalogo deve essere valido e correttamente identificato.<br>➢ La rimozione di un volo non deve compromettere la coerenza del sistema (es. prenotazioni già confermate). |
| Invarianti | // |

# Spiegazioni su:

### ❖ Pre-condizioni:

➢ Il **volo** non deve essere `null`.
➢ Il **volo** deve esistere nel catalogo dei voli del sistema ClickFly.

---

### ❖ Post-condizioni:

➢ Il **volo** non è più presente nel catalogo dei voli.
➢ Il metodo restituisce `true` se l'operazione di rimozione è avvenuta con successo, `false` altrimenti.

| Interfaccia | DataStorage(User) |
|---|---|
| Descrizione | `DataStorage(User)` gestisce la **memorizzazione** e il **recupero** delle informazioni relative agli utenti del sistema ClickFly, garantendo la persistenza dei dati necessari all'autenticazione, all'autorizzazione e alla gestione delle prenotazioni. |
| Metodi | `+saveUser(User u): boolean`<br><br>`+getUser(String email): User` |
| Invarianti di classi | Ogni **utente** deve essere identificabile tramite un'**email valida e univoca**.<br><br>Ogni **utente** deve possedere tutte le **informazioni personali obbligatorie** richieste dal sistema (es. email, password, dati anagrafici).<br><br>Ogni **utente** deve avere un **ruolo valido**, conforme ai ruoli definiti nel **dominio applicativo di ClickFly** (es. utente registrato, operatore di gestione). |

| Nome Metodo | +saveUser(User u): boolean |
| --- | --- |
| Descrizione | Questo metodo consente di **salvare un nuovo utente nel database di ClickFly**, memorizzandone i dati personali, le credenziali di accesso e il ruolo associato. L'operazione viene eseguita solo se l'utente non è già presente nel sistema. |
| Pre Condizioni | `context UserPersistence::saveUser(u: User): boolean`<br><br>**pre:**<br><br>• `u <> null`<br><br>• `not u.email.isEmpty()`<br><br>• Non deve esistere nel sistema un altro utente con la **stessa email**:<br><br>`User.allInstances()->forAll(existing \| existing.email <> u.email)`<br><br>• Il ruolo dell'utente deve essere valido e definito:<br>`u.role <> null` |
| Post Condizioni | `context UserPersistence::saveUser(u: User): boolean`<br><br>**post:**<br><br>• Se `result = true` allora l'utente viene salvato correttamente:<br><br>`User.allInstances()->includes(u)` |
| Invarianti | Ogni utente nel sistema ClickFly deve essere identificato da un'**email univoca**. |

| | |
|---|---|
| | Ogni utente deve avere un **ruolo assegnato**, coerente con quelli definiti nel dominio applicativo.<br><br>Non possono esistere due utenti con la stessa email all'interno del sistema. |

## Spiegazioni su:

**Pre-condizioni:**

L'oggetto **User** non deve essere nullo.
L'email dell'utente deve essere **valida e univoca** all'interno del sistema ClickFly.
Il ruolo dell'utente deve essere **specificato e coerente con i ruoli previsti dal dominio applicativo di ClickFly** (ad esempio *Utente registrato* o *Gestore prenotazioni*).

**Post-condizioni:**

Il metodo restituisce **true** se l'utente è stato salvato correttamente nel database di ClickFly.

| Nome Metodo | +getUser(String email): User |
|---|---|
| Descrizione | Questo metodo consente di **recuperare un utente registrato nel sistema ClickFly** a partire dall'indirizzo email fornito, restituendo l'oggetto `User` associato. |
| Pre Condizioni | **context UserPersistence::getUser(email: String): User**<br><br>`pre:`<br>`not email.isEmpty()`<br>`and`<br>`User.allInstances()->exists(u \|`<br>`u.email = email)` |
| Post Condizioni | **context UserPersistence::getUser(email: String): User**<br><br>`post:`<br>`result.email = email`<br>`and result.role <> null` |

| | |
|---|---|
| Invarianti | **context UserPersistence**<br><br>`inv:`<br>`User.allInstances()->forAll(u \|`<br>`  not u.email.isEmpty()`<br>`  and u.role <> null`<br>`)` |

## Spiegazioni

**Pre-condizioni:**

- L'email fornita **non deve essere vuota**.
- Deve **esistere un utente registrato nel sistema ClickFly** associato all'email specificata.

**Post-condizioni:**

- Il metodo restituisce **l'utente corrispondente all'email fornita**, comprensivo delle informazioni associate all'account.
- L'utente restituito **ha un ruolo valido** (ad esempio utente registrato o operatore di gestione), necessario per il corretto controllo degli accessi alle funzionalità del sistema.

| | |
|---|---|
| Interfaccia | DataStorage (Order) – ClickFly |
| Descrizione | La **DataStorage(Order)** gestisce la memorizzazione e il recupero dei dati relativi alle **prenotazioni dei voli** effettuate dagli utenti nel sistema ClickFly.<br>Si occupa della persistenza delle informazioni associate a una prenotazione, come i voli selezionati, lo stato della prenotazione e le date rilevanti. |
| Metodi | +saveOrder(Order o): boolean |
| Invarianti di Classe | • Ogni prenotazione deve essere associata a **un utente valido**.<br>• Ogni prenotazione deve contenere **almeno un volo** per essere |

| | |
|---|---|
| | considerata valida.<br>• Ogni prenotazione deve avere **uno stato valido** (ad esempio: *in elaborazione*, *confermata*, *annullata*).<br>• Le date associate alla prenotazione devono essere **coerenti** (la data di prenotazione non può essere successiva alla data del volo). |

| | |
|---|---|
| Nome Metodo | +saveOrder(Order o): boolean |
| Descrizione | Questo metodo consente di **salvare una prenotazione di volo** nel database del sistema ClickFly.<br>La prenotazione viene registrata solo se è valida e non esiste già una prenotazione con lo stesso identificativo. |
| Pre Condizioni | **context OrderPersistence::saveOrder(o: Order): boolean**<br>`pre:`<br>`o <> null`<br>`and not o.orderId.isEmpty()`<br>`and`<br>`Order.allInstances()->forAll(existing \| existing.orderId <> o.orderId)`<br>`and o.user <> null`<br>`and o.flights->notEmpty()` |
| Post Condizioni | **context OrderPersistence::saveOrder(o: Order): boolean**<br>`post:`<br>`result = true implies Order.allInstances()->includes(o)` |

| | |
|---|---|
| | **Spiegazione delle post-condizioni:**<br>• Se il metodo restituisce **true**, la prenotazione è stata correttamente salvata nel sistema.<br>• La prenotazione risulta persistita nel database. |
| Invarianti | **context OrderPersistence**<br>`inv:`<br>`Order.allInstances()->forAll(ord`<br>`\|`<br>`  ord.orderId <> null`<br>`  and not`<br>`ord.orderId.isEmpty()`<br>`  and ord.user <> null`<br>`  and ord.flights->notEmpty()`<br>`)` |

# Spiegazioni

## Pre-condizioni

L'oggetto **Order** (prenotazione di volo) non deve essere nullo.
L'identificativo della prenotazione (**orderId**) deve essere univoco all'interno del sistema ClickFly, ovvero non deve esistere un'altra prenotazione con lo stesso identificativo.

---

## Post-condizioni

Il metodo restituisce **true** se la prenotazione è stata correttamente salvata nel database del sistema ClickFly.

| **Nome Metodo** | +saveOrder(Order o): boolean |
|---|---|
| **Descrizione** | Questo metodo consente di **salvare una prenotazione di volo** nel database del sistema ClickFly.<br>La prenotazione viene registrata solo se è valida e non esiste già una prenotazione con lo stesso identificativo. |
| **Pre Condizione** | **context OrderPersistence::saveOrder(o: Order): boolean**<br><br>`pre:`<br>`o <> null`<br>`and not o.orderId.isEmpty()`<br>`and`<br>`Order.allInstances()->forAll(existing \| existing.orderId <> o.orderId)`<br>`and o.user <> null`<br>`and o.flights->notEmpty()` |
| **Post Condizione** | **context OrderPersistence::saveOrder(o: Order): boolean**<br><br>`post:`<br>`result = true implies Order.allInstances()->includes(o)` |
| **Invarianti** | **context OrderPersistence**<br><br>`inv:`<br>`Order.allInstances()->forAll(ord \|`<br>`  ord.orderId <> null`<br>`  and not ord.orderId.isEmpty()`<br>`  and ord.user <> null`<br>`  and ord.flights->notEmpty()`<br>`)` |

Spiegazioni Su:

Pre-condizioni:

Il carrello non deve essere nullo.

Il carrello deve essere associato un utente valido.

Post-condizioni:

Restituisce true se il carrello è stato salvato correttamente

| Interfaccia | DataStorage (Catalog) |
|---|---|
| Descrizione | La **DataStorage (Catalog)** gestisce la memorizzazione e il recupero dei dati relativi al **catalogo dei voli** disponibili nel sistema ClickFly.<br>Include informazioni quali dettagli del volo, prezzo, disponibilità dei posti, aeroporti di partenza e arrivo, date e compagnia aerea associata. |
| Metodi | +saveFlight(Flight f) : boolean |
| Invariante di classe | Ogni volo deve avere un **identificativo univoco e non vuoto**.<br><br>Ogni volo deve avere un **prezzo maggiore di zero**.<br><br>Ogni volo deve essere associato a:<br><br>● un **aeroporto di partenza valido**,<br>● un **aeroporto di arrivo valido**,<br>● una **compagnia aerea esistente**.<br><br>Ogni volo deve specificare **data e ora di partenza e di arrivo** valide.<br><br>La disponibilità dei posti di un volo non può |

| | |
|---|---|
| | mai essere negativa. |

| | |
|---|---|
| Nome Metodo | +saveFlight(Flight f) : boolean |
| Descrizione | Questo metodo consente di **salvare un volo** nel sistema ClickFly, rendendolo disponibile nel catalogo dei voli per la ricerca e la prenotazione da parte degli utenti. |
| Pre Condizione | context FlightsPersistence::saveFlight(f : Flight) : boolean<br>pre:<br>f <> null<br>and not f.flightId.isEmpty()<br>and f.price > 0<br>and f.departureAirport <> null<br>and f.arrivalAirport <> null<br>and f.departureDateTime > currentDateTime<br>and Flight.allInstances()->forAll(existing \| existing.flightId <> f.flightId) |
| Post Condizione | context FlightsPersistence::saveFlight(f : Flight) : boolean<br>post:<br>result = true implies Flight.allInstances()->includes(f) |
| Invarianti | context FlightsPersistence<br>inv:<br>Flight.allInstances()->forAll(f \|<br>not f.flightId.isEmpty()<br>and f.price > 0<br>and f.departureAirport <> null<br>and f.arrivalAirport <> null<br>) |

# Spiegazioni

## ❖ Pre-condizioni:

➢ L'oggetto **Flight** non deve essere nullo.
➢ Il volo deve avere un **identificativo univoco** (`flightId`) che lo distingua da tutti gli altri voli presenti nel sistema.
➢ Il volo deve avere un **prezzo maggiore di zero**, in quanto non sono ammessi voli con costo nullo o negativo.
➢ Il volo deve avere **aeroporto di partenza e di arrivo validi**.
➢ La **data e l'orario di partenza** devono essere successivi al momento corrente.

## ❖ Post-condizioni:

➢ Il metodo restituisce **true** se il volo è stato **salvato correttamente nel sistema** ed è ora disponibile nel catalogo dei voli di ClickFly.

| **Interfaccia** | UserView |
|---|---|
| **Descrizione** | `serView` gestisce le operazioni di interazione tra l'utente e il sistema ClickFly relative alla **registrazione**, **autenticazione** e **visualizzazione dei dati dell'utente**. Fornisce i servizi necessari per permettere agli utenti di creare un account, accedere al sistema e recuperare le informazioni del proprio profilo. |
| **Metodi** | + register(String email, String username, String password1, String password2): User<br>+ getUser(String email): User |
| **Invariante di classe** | • Ogni utente deve essere identificato da un'**email valida e univoca**.<br>• Le password dell'utente devono essere **memorizzate esclusivamente in forma hashata**.<br>• Ogni utente deve avere un **ruolo valido** appartenente al dominio applicativo di ClickFly (es. User, GestoreOrdini). |

| | |
|---|---|
| | • Non possono esistere due utenti con la **stessa email** nel sistema. |

| | |
|---|---|
| **Nome Metodo** | + register(String email, String username, String password1, String password2): User |
| **Descrizione** | Questo metodo permette la **registrazione di un nuovo utente** sulla piattaforma ClickFly, creando un account associato all'email fornita e inizializzando correttamente le credenziali di accesso. |
| **Pre Condizioni** | context UserView::register(<br>email: String,<br>username: String,<br>password1: String,<br>password2: String<br>): User<br>pre:<br>not email.isEmpty()<br>and not username.isEmpty()<br>and password1 = password2<br>and User.allInstances()->forAll(u \| u.email <> email) |
| **Post Condizioni** | context UserView::register(<br>email: String,<br>username: String,<br>password1: String,<br>password2: String<br>): User<br>post:<br>result <> null<br>and result.email = email<br>and result.username = username<br>and result.hashedPassword = hash(password1)<br>and result.role = 'User' |
| **Invarianti** | inv: |

| | |
|---|---|
| | not result.email.isEmpty()<br>and result.role <> null<br>and not result.hashedPassword.isEmpty() |

**Spiegazioni su:**

**Pre-condizioni:**

- L'email non deve essere già associata a un altro account.
- Il nome utente(username)non deve essere già utilizzato.
- La password 1 deve essere uguale alla password 2.

**Post-condizioni:**

- Restituisce l'utente registrato se l'operazione ha successo

| | |
|---|---|
| Nome Metodo | + getUser(String email): User |
| Descrizione | Questo metodo permette di **recuperare un utente registrato nel sistema ClickFly** a partire dall'indirizzo email fornito. |
| Pre Condizione | context UserPersistence::getUser(email: String): User<br>pre:<br>not email.isEmpty()<br>and User.allInstances()->exists(u \| u.email = email) |
| Post Condizione | context UserPersistence::getUser(email: String): User<br>post:<br>result <> null<br>and result.email = email<br>and result.role <> null |
| Invarianti | inv:<br>not email.isEmpty()<br>and result.email = email<br>and result.role <> null |

| | |
|---|---|

La pre-condizione `not email.isEmpty()` verifica che l'indirizzo email fornito non sia vuoto, poiché l'email rappresenta l'identificativo univoco dell'utente nel sistema ClickFly.

➢ La condizione `User.allInstances()->exists(u | u.email = email)` assicura che esista almeno un utente registrato nel sistema ClickFly con l'email fornita, evitando il recupero di utenti inesistenti.

| Interfaccia | CatalogView |
|---|---|
| Descrizione | CatalogView gestisce la visualizzazione del catalogo dei voli disponibili su ClickFly, fornendo supporto alla ricerca e al filtraggio dei voli in base a diversi criteri, come destinazione, compagnia aerea e prezzo. L'interfaccia consente agli utenti (guest e registrati) di consultare i voli senza modificare i dati sottostanti. |
| Metodi | `+getFlights(): Set<Volo>`<br><br>`+getFlightById(String id): Volo`<br><br>`+getFlightsByAirline(String airline): Set<Volo>`<br><br>`+getFlightsByPrice(boolean ascending): Set<Volo>`<br><br>`+getFlightsByAirlineAndPrice(String airline, boolean ascending): Set<Volo>` |
| Invariante di classe | Il catalogo dei voli non può essere vuoto.<br><br>Gli identificativi dei voli (ID volo) devono essere univoci.<br><br>Il prezzo di ogni volo deve essere un valore positivo. |

| Nome Metodo | +getFlights(): Set<Volo> |
|---|---|
| Descrizione | Questo metodo permette di recuperare l'insieme di tutti i voli disponibili presenti nel catalogo di ClickFly, rendendoli accessibili per la visualizzazione e la ricerca da parte degli utenti. |
| Pre Condizioni | (Non sono richieste pre-condizioni: il metodo può essere invocato da qualsiasi utente, autenticato o guest) |
| Post Condizioni | context CatalogView::getFlights(): Set(Volo)<br>post:<br>result->notEmpty() |
| Invarianti | Il catalogo dei voli deve contenere solo voli validi.<br><br>Ogni volo presente nel risultato deve avere:<br><br>● un identificativo univoco,<br>● un prezzo maggiore di zero,<br>● una data di partenza valida.<br><br>Il metodo restituisce sempre una collezione coerente con lo stato attuale del catalogo. |

Spiegazioni su:

Pre-condizioni:

Nessuna

Post-condizioni:

Restituisce l'insieme di tutti i prodotti nel catalogo

| Nome Metodo | +getFlightById(String id): Volo |
|---|---|
| Descrizione | Questo metodo permette di ottenere un volo specifico dal catalogo di ClickFly a partire dal suo identificativo univoco. |
| Pre Condizione | context CatalogView::getFlightById(id: String): Volo<br>pre:<br>not id.isEmpty()<br>and Volo.allInstances()->exists(v \| v.id = id) |
| Post Condizione | context CatalogView::getFlightById(id: String): Volo<br>post:<br>result.id = id |
| Invarianti | L'identificativo del volo non deve essere vuoto.<br><br>Ogni volo nel sistema deve avere un identificativo univoco.<br><br>Il volo restituito deve appartenere al catalogo dei voli disponibili.<br><br>Il metodo non modifica lo stato del sistema (operazione di sola lettura). |

## Spiegazioni

❖ **Pre-condizioni:**

- L'ID del **volo** deve essere valido e non vuoto.
- Deve esistere nel sistema un volo associato all'ID fornito.

❖ **Post-condizioni:**

- Il sistema restituisce il **volo** corrispondente all'ID specificato, se presente nel catalogo dei voli.

| | |
|---|---|
| **Nome Metodo** | +getFlightsByAirline(String airline): Set<Volo> |
| **Descrizione** | Questo metodo permette di ottenere una collezione di **voli** associati a una determinata **compagnia aerea**. |
| **Pre Condizioni** | context CatalogView::getFlightsByAirline(airline: String): Set(Volo)<br>pre:<br>not airline.isEmpty()<br>and Volo.allInstances()->exists(v \| v.compagniaAerea = airline) |
| **Post Condizioni** | context CatalogView::getFlightsByAirline(airline: String): Set(Volo)<br>post:<br>result->forAll(v \| v.compagniaAerea = airline) |
| **Invarianti** | • Ogni volo restituito deve appartenere alla compagnia aerea specificata.<br>• Il catalogo dei voli contiene solo voli validi.<br>• Ogni volo è identificato univocamente da un ID. |

## Spiegazioni

❖ **Pre-condizioni:**

➢ Il nome della **compagnia aerea** deve essere valido e non vuoto.

❖ **Post-condizioni:**

➢ Restituisce un insieme di **voli** appartenenti alla **compagnia aerea** specificata.

| | |
|---|---|
| Nome Metodo | +getFlightsByPrice(boolean ascending): Set<Volo> |
| Descrizione | Questo metodo permette di ottenere una collezione di **voli** ordinati in base al **prezzo**, secondo il criterio specificato (crescente o decrescente). |
| Pre Condizione | Nessuna pre-condizione specifica.<br>Il catalogo dei voli può essere vuoto o contenere uno o più voli. |
| Post Condizione | `contextCatalogView::getFlightsByPrice(ascending: Boolean): Set(Volo)`<br><br>`post:`<br>`if ascending`<br>`then`<br>`  result = Volo.allInstances()->sortedBy(v \| v.prezzo)`<br>`else`<br>`  result = Volo.allInstances()->sortedBy(v \| v.prezzo)->reverse()`<br>`endif` |
| Invarianti | Ogni **volo** restituito deve avere un **prezzo positivo**.<br><br>L'ordinamento deve rispettare il criterio indicato dal parametro `ascending`.<br><br>Il metodo **non modifica** il catalogo dei voli, ma ne restituisce solo una vista ordinata. |

## Spiegazioni su:

❖ **Pre-condizioni:**

➢ Il criterio di ordinamento per prezzo deve essere valido:
il parametro `ascending` deve indicare un ordinamento **crescente** (`true`) oppure **decrescente** (`false`).

---

❖ **Post-condizioni:**

➢ Il metodo restituisce una collezione di **voli** ordinati in base al **prezzo del volo**.
➢ L'ordinamento è:

- **crescente** se `ascending = true`;
- **decrescente** se `ascending = false`.

| | |
|---|---|
| **Nome Metodo** | +getFlightsByCompanyAndPrice(String company, boolean ascending): Set<Volo> |
| **Descrizione** | Questo metodo permette di ottenere una collezione di **voli** filtrati per **compagnia aerea** e ordinati in base al **prezzo del volo**. |
| **Pre Condizione** | **context CatalogView::getFlightsByCompanyAndPrice(company: String, ascending: Boolean): Set(Volo)**<br><br>`pre:`<br>`not company.isEmpty()`<br>`and`<br>`Flight.allInstances()->exists(f | f.company = company)` |
| **Post Condizioni** | **context CatalogView::getFlightsByCompanyAndPrice(company: String, ascending: Boolean): Set(Volo)**<br><br>`post:`<br>`result->forAll(f | f.company = company)`<br>`and`<br>`if ascending`<br>`then`<br>`  result = result->sortedBy(f | f.price)` |

| | |
|---|---|
| | `else`<br>`    result = result->sortedBy(f | f.price)->reverse()`<br>`endif` |
| **Invarianti** | inv:<br>result->forAll(f \| f.price > 0)<br>and<br>result->forAll(f \| not f.company.isEmpty()) |

## Spiegazioni su:

❖ **Pre-condizioni:**

➢ La **compagnia aerea** deve essere valida e non vuota.
➢ Il criterio di **ordinamento per prezzo** deve essere valido (ascendente o discendente).

❖ **Post-condizioni:**

➢ Il metodo restituisce l'insieme dei **voli appartenenti alla compagnia aerea specificata**, ordinati per **prezzo** in ordine **crescente o decrescente** in base al valore del parametro `ascending`.

| Interfaccia | OrderView |
|---|---|
| Descrizione | OrderView gestisce la **visualizzazione delle prenotazioni dei voli** effettuate dagli utenti e consente agli **operatori di gestione** di monitorare e aggiornare lo stato delle prenotazioni presenti nel sistema. |
| Metodi | `+getOrderById(String orderId) : Order`<br><br>`+getOrdersByUser(String email) : Set<Order>`<br><br>`+getAllOrders() : Set<Order>` |

| | |
|---|---|
| Invariante di classi | L'**ID della prenotazione** deve essere **univoco** e **non nullo**.<br><br>Ogni prenotazione deve essere associata a un **utente valido**.<br><br>Ogni prenotazione deve essere associata ad almeno un **volo valido**. |

| | |
|---|---|
| Nome Metodo | +getOrderById(String orderId) : Order |
| Descrizione | Questo metodo permette di ottenere una **prenotazione di volo** a partire dal suo **identificativo univoco**. |
| Pre Condizione | **context OrderView::getOrderById(orderId : String) : Order**<br><br>`pre:`<br>`not orderId.isEmpty()`<br>`and`<br>`Order.allInstances()->exists(o \|`<br>`o.id = orderId)` |
| Post Condizione | **context OrderView::getOrderById(orderId : String) : Order**<br><br>`post:`<br>`result.id = orderId` |
| Invarianti | Ogni prenotazione deve avere un **identificativo univoco e non nullo**.<br><br>La prenotazione restituita deve essere associata a un **utente valido**.<br><br>La prenotazione deve riferirsi a **un volo esistente nel sistem** |

## Spiegazioni

❖ **Pre-condizioni:**

- ➢ L'**ID della prenotazione** deve essere valido e non vuoto.
- ➢ Deve esistere nel sistema **una prenotazione associata all'ID fornito**.

❖ **Post-condizioni:**

- ➢ Il sistema restituisce la **prenotazione di volo** corrispondente all'ID specificato, se presente.

| | |
|---|---|
| **Nome Metodo** | `+getOrdersByUser(String email): Set<Order>` |
| **Descrizione** | Questo metodo permette di ottenere l'insieme delle **prenotazioni di voli** associate a un determinato utente, identificato tramite l'indirizzo email. |
| **Pre Condizioni** | **context OrderView::getOrdersByUser(email: String): Set(Order)**<br>`pre:`<br>`not email.isEmpty()`<br>`and`<br>`User.allInstances()->exists(u \| u.email = email)` |
| **Post Condizioni** | **context OrderView::getOrdersByUser(email: String): Set(Order)**<br>`post:`<br>`result->forAll(o \| o.user.email = email)` |
| **Invarianti** | Ogni prenotazione restituita deve essere |

| | |
|---|---|
| | **associata a un utente valido**.<br><br>Le prenotazioni visualizzate devono appartenere **esclusivamente all'utente autenticato**.<br><br>L'email dell'utente associato a ciascuna prenotazione non deve essere vuota. |

## Spiegazioni

❖ **Pre-condizioni:**

➢ L'email fornita deve essere valida e non vuota.
➢ Deve esistere un utente registrato nel sistema ClickFly associato all'email specificata.

❖ **Post-condizioni:**

➢ Il metodo restituisce un insieme di **prenotazioni di voli** associate all'utente identificato dall'email fornita.
➢ Tutti gli ordini restituiti devono appartenere esclusivamente all'utente specificato.

| | |
|---|---|
| **Nome Metodo** | +getAllOrders(): Set<Order> |
| **Descrizione** | Questo metodo permette di recuperare **tutte le prenotazioni di voli presenti nel sistema ClickFly**.<br>È destinato alle figure di gestione degli ordini e consente la visualizzazione globale delle prenotazioni registrate. |
| **Pre Condizioni** | *Nessuna pre-condizione esplicita.*<br>Il metodo è invocabile solo da componenti autorizzati (es. gestore ordini), come stabilito dal controllo degli accessi a livello di sistema. |
| **Post Condizioni** | **context OrderView::getAllOrders(): Set(Order)**<br><br>`post:`<br>`result->notEmpty()` |

| | |
|---|---|
| | ➢ Il risultato contiene l'insieme di **tutte le prenotazioni** registrate nel sistema ClickFly. |
| **Invarianti** | Ogni ordine restituito deve avere un **identificativo univoco**.<br>➢ Ogni ordine deve essere associato a un **utente valido**.<br>➢ Ogni ordine deve avere uno **stato valido** (es. "in elaborazione", "confermato", "annullato"). |

| | |
|---|---|
| **Nome Metodo** | +getAllOrders(): Set<Order> |
| **Descrizione** | Questo metodo permette di recuperare **tutte le prenotazioni di voli presenti nel sistema ClickFly**.<br>È destinato alle figure di gestione degli ordini e consente la visualizzazione globale delle prenotazioni registrate. |
| **Pre Condizione** | *Nessuna pre-condizione esplicita.*<br>Il metodo è invocabile solo da componenti autorizzati (es. gestore ordini), come stabilito dal controllo degli accessi a livello di sistema. |
| **Post Condizione** | **context OrderView::getAllOrders(): Set(Order)**<br><br>`post:`<br>`result->notEmpty()`<br><br>Il risultato contiene l'insieme di **tutte le prenotazioni** registrate nel sistema ClickFly. |
| **Invariante** | Ogni ordine restituito deve avere un **identificativo univoco**.<br>Ogni ordine deve essere associato a un **utente valido**. |

| | |
|---|---|
| | Ogni ordine deve avere uno **stato valido** (es. “in elaborazione”, “confermato”, “annullato”). |

## Spiegazioni su:

❖ **Pre-condizioni:**

➢ Non è richiesta alcuna pre-condizione specifica per l’esecuzione del metodo.
➢ L’accesso al metodo è regolato dal sistema di controllo degli accessi, che ne consente l’utilizzo esclusivamente agli utenti autorizzati (es. gestore ordini).

❖ **Post-condizioni:**

➢ Il metodo restituisce l’insieme di **tutte le prenotazioni di voli presenti nel sistema ClickFly**.
➢ Ogni ordine restituito contiene informazioni coerenti e valide, come utente associato, voli prenotati, stato e totale.

| **Interfaccia** | CartView |
|---|---|
| **Descrizione** | CartView gestisce la **visualizzazione e l’accesso ai dati del carrello voli** associato a un utente del sistema ClickFly. Consente all’utente autenticato di visualizzare i voli selezionati prima della conferma della prenotazione. |
| **Metodi** | + getCartByUser(String email) : Cart |
| **Invariante di classe** | Ogni carrello deve essere **associato a un utente valido e autenticato**.<br><br>Un carrello non può esistere senza un utente proprietario.<br><br>Ogni carrello contiene esclusivamente voli selezionati dall’utente associato. |

| **Nome Metodo** | + getCartByUser(String email) : Cart |
|---|---|

| | |
|---|---|
| **Descrizione** | Questo metodo permette di **recuperare il carrello voli associato a un utente**, identificato tramite la sua email.<br>Il carrello contiene i voli selezionati dall'utente prima della conferma della prenotazione. |
| **Pre Condizione** | context CartView::getCartByUser(email: String) : Cart<br>pre:<br>not email.isEmpty()<br>and User.allInstances()->exists(u \| u.email = email)<br><br>L'email fornita non deve essere vuota.<br>Deve esistere un utente registrato nel sistema con l'email specificata. |
| **Post Condizione** | context CartView::getCartByUser(email: String) : Cart<br>post:<br>result <> null<br>and result.user.email = email<br>Il metodo restituisce un carrello valido.<br><br>Il carrello restituito è associato all'utente identificato dall'email fornita. |
| **Invariante** | inv:<br>result.user.email = email<br>and result.user <> null<br>Ogni carrello è sempre associato a un utente valido.<br><br>Non può esistere un carrello senza un utente proprietario. |

## Spiegazioni su:

❖ **Pre-condizioni:**

➢ L'email fornita deve essere valida e non vuota.
➢ Deve esistere un utente registrato nel sistema ClickFly associato all'email specificata.

❖ **Post-condizioni:**

➢ Il metodo restituisce il carrello voli associato all'utente identificato dall'email fornita, contenente le eventuali prenotazioni selezionate ma non ancora confermate.

# Design Patterns

Per garantire una progettazione **robusta, modulare, scalabile e facilmente manutenibile**, nel sistema **ClickFly** sono stati adottati i seguenti design pattern architetturali e strutturali.

---

## 4.1 Model–View–Controller (MVC)

**Motivo**
Il pattern MVC è stato adottato per separare in modo chiaro:

- la **logica di presentazione** (View),
- la **logica applicativa e di controllo** (Controller),
- la **gestione dei dati e delle entità di dominio** (Model).

Questa separazione consente una maggiore manutenibilità del codice, facilita l'evoluzione del sistema e riduce le dipendenze tra componenti.

**Applicazione in ClickFly**

- **Model**
  Include le classi che rappresentano il dominio applicativo e l'accesso ai dati, tra cui:
  - `Volo`
  - `Prenotazione`
  - `Carrello`
  - `Utente`
  - DAO (`VoloDAO`, `PrenotazioneDAO`, `UtenteDAO`)
- **View**
  Comprende le pagine JSP e le classi di visualizzazione responsabili dell'interazione con l'utente, ad esempio:

    - `CatalogView` (visualizzazione e ricerca voli)
    - `CartView` (visualizzazione carrello)
    - `OrderView` (visualizzazione prenotazioni)
    - `UserView` (registrazione, login, profilo)

- **Controller**
  Implementato tramite Servlet e Service Layer, gestisce le richieste HTTP e coordina la logica applicativa:
    - `AuthenticationService`
    - `CatalogService`
    - `CartService`
    - `OrderService`

---

## 4.2 DAO (Data Access Object)

**Motivo**
Il pattern DAO è stato utilizzato per separare la **logica di accesso ai dati** dalla **logica di business**, rendendo il sistema indipendente dal DBMS utilizzato e facilitando eventuali modifiche future alla persistenza.

**Applicazione in ClickFly**

- Le classi DAO incapsulano tutte le operazioni di lettura e scrittura sul database MySQL, tra cui:
    - `VoloDAO`
    - `PrenotazioneDAO`
    - `UtenteDAO`
    - `CarrelloDAO`
- Ogni DAO fornisce metodi specifici per:

    - recuperare voli disponibili,
    - salvare e aggiornare prenotazioni,
    - gestire utenti e sessioni,
    - garantire coerenza e integrità dei dati.

---

### 4.3 Strategy (implicito)

**Motivo**
Il pattern Strategy consente di gestire comportamenti variabili senza modificare il codice esistente.

**Applicazione in ClickFly**

- Gestione di **diverse modalità di pagamento**, ad esempio:
    - pagamento tramite carta di credito,
    - pagamento tramite portafoglio virtuale.
- Possibile estensione futura per:
    - politiche di rimborso differenti,
    - tariffe dinamiche,
    - classi di volo (economy, business, first class).

# Glossario

**Autenticazione**
Processo di verifica dell'identità di un utente tramite credenziali, come email e password, per consentire l'accesso alle funzionalità del sistema ClickFly.

**Autorizzazione**
Processo che verifica se un utente autenticato possiede i permessi necessari per eseguire una determinata azione (es. prenotare un volo, modificare una prenotazione).

**Cache locale**
Memorizzazione temporanea di dati (es. risultati di ricerca voli o dati di sessione) vicino al punto di utilizzo per migliorare le prestazioni del sistema.

**Caching**
Tecnica che consente di memorizzare temporaneamente dati frequentemente utilizzati, riducendo l'accesso al database e migliorando i tempi di risposta.

**Carrello**
Struttura temporanea che contiene i voli selezionati da un utente prima della conferma della prenotazione.

**Collezione**
Struttura dati che memorizza più elementi (es. liste, set), utilizzata per gestire voli, prenotazioni o risultati di ricerca.

**Configurazione**
Insieme di impostazioni e parametri che definiscono il comportamento del sistema ClickFly (es. connessione al database, gestione sessioni).

**Data Access Object (DAO)**
Pattern architetturale che separa la logica di accesso ai dati dalla logica applicativa, utilizzato per la gestione di utenti, voli e prenotazioni.

**Data Transfer Object (DTO)**
Oggetto utilizzato per trasferire dati tra i diversi layer del sistema (Presentation, Service, Persistence).

**Eccezione di runtime**
Errore che si verifica durante l'esecuzione del programma (es. errore di pagamento o indisponibilità posti) e che richiede una gestione immediata.

**Eccezioni**
Meccanismo utilizzato per intercettare e gestire errori o comportamenti inattesi durante l'esecuzione del codice.

**Framework**
Struttura software che fornisce funzionalità generali per semplificare lo sviluppo dell'applicazione (es. Bootstrap per il front-end).

**Gestione errori**
Insieme di strategie adottate per intercettare, segnalare e gestire situazioni anomale, garantendo la stabilità del sistema.

**Hashing**
Processo di conversione di dati sensibili (come le password) in una stringa univoca e non reversibile, utilizzato per garantire la sicurezza.

**Invarianti**
Condizioni che devono essere sempre vere durante il ciclo di vita di un oggetto o di una classe (es. una prenotazione deve essere associata a un utente valido).

**Interfaccia**
Contratto che definisce i metodi pubblici e il comportamento di una classe o di un servizio.

**Layer**
Livello logico dell'architettura software che separa responsabilità specifiche (Presentation Layer, Service Layer, Persistence Layer).

**Metodo**
Funzione associata a un oggetto o a una classe che può essere invocata per eseguire una specifica operazione.

**Model-View-Controller (MVC)**
Pattern architetturale che separa la logica di presentazione (View), la logica applicativa (Controller) e la gestione dei dati (Model).

**Pattern di progettazione**
Soluzione generica e riutilizzabile per problemi ricorrenti nella progettazione del software.

**Post-condizioni**
Condizioni che devono essere vere dopo l'esecuzione di un metodo o di una funzione.

**Pre-condizioni**
Condizioni che devono essere vere prima dell'esecuzione di un metodo o di una funzione.

**Prenotazione**
Entità che rappresenta l'acquisto confermato di uno o più posti su un volo da parte di un utente.

**Query**
Richiesta di informazioni o operazioni effettuata su un database (es. ricerca voli disponibili).

**Ruoli**
Permessi o responsabilità assegnati agli utenti del sistema (es. utente registrato, gestore ordini).

**Sessione**
Contesto attivo che rappresenta un utente autenticato e le sue interazioni con il sistema ClickFly.

**Set**
Struttura dati che memorizza un insieme di elementi unici, senza duplicati (es. insieme di voli).

**Sicurezza**
Insieme di misure e tecniche adottate per proteggere dati sensibili e accessi da utilizzi non autorizzati.

**Validazione**

Processo che verifica che i dati forniti dall’utente rispettino criteri e requisiti definiti dal sistema (es. formato email, disponibilità posti).