UNIVERSITÀ DEGLI STUDI DI NAPOLI

FEDERICO II

DOCUMENTAZIONE PER PROGETTO OBJECT-ORIENTED

CdL Triennale in Informatica

Corso Object-Oriented

Rocco Molinari
N86005071

Mirko Maiello
N86005102

Andrea Marquez
N86005189

Anno Accademico: 2024/2025

Indice

1 Introduzione

1.1 Traccia del progetto e analisi dei requisiti

Si sviluppi un sistema informativo per la gestione dell'aeroporto di Napoli, composto da una base di dati relazionale e da un applicativo Java con interfaccia grafica realizzata con Swing. Questo sistema deve consentire di organizzare e monitorare le operazioni aeroportuali in modo efficiente e intuitivo.

Il sistema può essere utilizzato da utenti autenticati tramite una login e una password. Gli utenti sono suddivisi in due ruoli: utenti generici, che possono prenotare voli, e amministratori del sistema, che gestiscono l'inserimento e l'aggiornamento dei voli.

Il sistema gestisce i voli in arrivo e quelli in partenza. Ogni volo è caratterizzato da un codice univoco, la compagnia aerea, l'aeroporto di origine (per i voli in arrivo a Napoli) e quello di destinazione (per i voli in partenza da Napoli), la data del volo, l'orario previsto, l'eventuale ritardo e lo stato del volo (programmato, decollato, in ritardo, atterrato, cancellato). Gli amministratori del sistema hanno la possibilità di inserire nuovi voli e aggiornare le informazioni sui voli esistenti.

Gli utenti generici possono effettuare prenotazioni per i voli programmati. Ogni prenotazione è legata a un volo e contiene informazioni come i dati del passeggero (che non deve necessariamente coincidere con il nome dell'utente che lo ha prenotato), il numero del biglietto, il posto assegnato e lo stato della prenotazione (confermata, in attesa, cancellata). Gli utenti possono cercare e modificare le proprie prenotazioni in base al nome del passeggero o al numero del volo.

Il sistema gestisce anche i gate di imbarco (identificati da un numero), assegnandoli ai voli in partenza. Gli amministratori possono modificare l'assegnazione dei gate.

Il sistema consente agli utenti di visualizzare aggiornamenti sui voli prenotati accedendo alla propria area personale, dove possono controllare eventuali ritardi, cancellazioni o variazioni direttamente dall'interfaccia. Inoltre, all'interno della homepage degli utenti viene mostrata una tabella con gli orari aggiornati dei voli in partenza e in arrivo, fornendo una panoramica immediata delle operazioni aeroportuali.

Infine, il sistema permette di eseguire ricerche rapide per trovare voli, passeggeri e bagagli in base a diversi criteri. Le informazioni più importanti vengono evidenziate, come i voli in ritardo o cancellati, per facilitare la gestione delle operazioni aeroportuali.

Requisito per gruppi da tre persone:

Un'altra funzione importante è il monitoraggio dei bagagli. Ogni bagaglio è associato al volo del passeggero e viene registrato nel sistema durante l'operazione di check-in, generando un codice univoco che consente il tracciamento. Durante il percorso, lo stato del bagaglio viene gestito manualmente dagli amministratori, che aggiornano il sistema indicando se è stato caricato sull'aereo o è disponibile per il ritiro. Gli utenti possono visualizzare lo stato aggiornato del proprio bagaglio tramite l'interfaccia del sistema. Se un bagaglio viene smarrito, l'utente può segnalarlo direttamente nel sistema. Gli amministratori possono accedere a un modulo dedicato per visionare le richieste di smarrimento.

2 Package Model

2.1 Classi principali e attributi

il package model progettato è stato realizzato con scelte strutturali e logiche finalizzate a soddisfare i requisiti funzionali e informativi del sistema di gestione aeroportuale. Il modello supporta le operazioni di prenotazione, gestione voli, monitoraggio bagagli e ruoli utente differenziati.

Classi: Utente, UtenteGenerico, Amministratore

Motivazione Progettuale:

Questa scelta nasce dall'esigenza di modellare in modo chiaro i diversi protagonisti del sistema. La superclasse Utente consente di gestire le funzionalità comuni a tutti i profili, come l'autenticazione. La specializzazione in Utente Generico e Ammini- stratore riflette la necessità di distinguere i permessi e le operazioni disponibili per ciascun tipo di utente, semplificando la gestione dei ruoli.

Attributi comuni (Utente):

- login
- password

Classi: Volo, VoloInArrivo, VoloInPartenza, StatoVolo

Motivazione Progettuale:

La classe Volo è un'entità centrale nel sistema, e la sua rappresentazione generalizzata consente di mantenere coerenza tra le informazioni comuni. Tuttavia, optare per questa scelta progettuale e usufruire di due sottoclassi ci ha dato la possibilità di avere una certa eterogeneità operativa tra voli in arrivo e in partenza (ad esempio, l'assegnazione dei gate è rilevante solo per le partenze). L'enumerazione Stato Volo è stata introdotta per poter tenere costantemente traccia del ciclo di vita del volo.

Attributi Volo:

- codiceVolo, compagnia, data, orarioPrevisto, ritardo
- statoVolo, origine, destinazione

Specializzazioni e relativi Attributi:

- VoloInArrivo: destinazione = "Napoli"
- VoloInPartenza: aeroportoDestinazione, origine = "napoli", gate

Valori StatoVolo:

- programmato, decollato, inRitardo, cancellato

Classi: Prenotazione, StatoPrenotazione

Motivazione Progettuale: La classe Prenotazione è stata introdotta per modellare in maniera puntuale il processo di accaparramento dei voli, fornendo tutte le informazioni necessarie alla gestione e al tracciamento del biglietto. La separazione tra Utente e Passeggero consente di coprire scenari reali in cui chi prenota non è necessariamente chi viaggia, migliorando la flessibilità e la precisione del sistema. L'enumerazione Stato Prenotazione supporta una gestione strutturata dello stato delle prenotazioni (confermate, in attesa, cancellate), semplificando il monitoraggio e le modifiche.

Attributi Prenotazione:

- numeroBiglietto, postoAssegnato, statoPrenotazione

Attributi Passeggero:

- idDocumento, nome, cognome, dataNascita

Valori StatoPrenotazione:

- confermata, inAttesa, cancellata

Classe: Passeggero

Motivazione Progettuale: La separazione tra Utente e Passeggero consente di coprire scenari reali in cui chi prenota non è necessariamente chi viaggia, migliorando la flessibilità e la precisione del sistema.

Attributi Passeggero:

- idDocumento, nome, cognome, dataNascita

Classi: Bagaglio, StatoBagaglio

Motivazione Progettuale: La classe Bagaglio permette di gestire ogni singolo bagaglio in relazione al volo e alla prenotazione garantendo tracciabilità e controllo durante le fasi del viaggio. La gestione dello stato del bagaglio tramite enumerazione Stato Bagaglio chiarisce il suo stato, se è disponibile o caricato.

Attributi Bagaglio:

- codiceBagaglio, statoBagaglio

Valori StatoBagaglio:

- caricato, ritirabile

2.2 Associazioni

Prenotazione (UtenteGenerico <-> Volo)

in questa associazione si fa uso della classe associativa "Prenotazione"

Associazione N:N

- Un utente generico può avere da 0 a N prenotazioni
- Un volo può avere da 0 a N prenotazioni

Appartiene (Bagaglio -> Prenotazione)

Associazione 1:N

- Una prenotazione può avere da 0 a N bagagli
- Ogni bagaglio è associato a una sola prenotazione

Possiede (Passeggero -> Prenotazione)

Associazione 1:N

- Una prenotazione è posseduta da un solo passeggero
- Un passeggero possiede da 1 a N prenotazioni

3 Package GUI

Il progetto si propone di fornire un sistema integrato per la gestione degli accessi e delle registrazioni degli utenti, dei voli (in partenza e in arrivo), dei bagagli ed eventuali segnalazioni associati. I protagonisti del progetto si suddividono in: utente generico e amministratore. L'interazione con il sistema avviene tramite una serie di interfacce grafiche (GUI), sviluppate nel package gui, che si interfacciano con la logica di controllo situata nel package controller. Il sistema è stato progettato seguendo il pattern architetturale MVC (Model-View-Controller), per garantire una chiara separazione tra la logica della gestione dei dati e la visualizzazione.

Il package gui contiene tutte le classi dedicate alla gestione dell'interfaccia grafica tramite Swing.

3.1 Classi Principali e attributi

Classe DashBoardUser:

Questa classe implementa l'interfaccia grafica dedicata all'utente generico. Permette l'interazione con le funzionalità principali: visualizzazione e prenotazione voli, gestione bagagli, ecc. L'interazione con la GUI viene inoltrata al controller tramite eventi, mantenendo separata la logica di controllo dalla visualizzazione.

Attributi principali:

- **JPanel homePage:** pannello principale contenente l'intera dashboard utente.
- **Jpanel welcomePanel:** pannello dedicato al messaggio di benvenuto.
- **Jpanel welcomeText:** pannello dedicato al messaggio di benvenuto.
- **JButton logoutUser:** pulsante per effettuare il logout o chiudere l'applicazione.
- **JLabel welcomeTextLabel:** etichetta che mostra ID o username dell'utente.
- **JComboBox comboBox1:** menù a tendina con le azioni selezionabili dall'utente.
- **JLabel outputLabel:** etichetta nascosta che mostra messaggi o risultati dinamici.
- **String username:** username dell'utente per personalizzare l'interfaccia.
- **JDialog choiceDialog:** Dialog modale per finestre secondarie.
- **Controller controller:** riferimento al controller.

Metodi principali:

- **DashBoardUser(String username, Controller controller):** costruttore che inizializza i componenti grafici, imposta la lista delle azioni nella combo box e associa i listener agli eventi generati dall'utente
- **actionPerformed(ActionEvent e):** gestisce gli eventii legati al pulsante di logout
- **itemStateChanged(ItemEvent e):** intercetta la selezione nella combo box e passa la scelta al controller tramite il metodo selectedItem(String, DashBoardUser)
- getter e setter per i componenti principali, utili per interagire con l'interfaccia da altre classi o dal controller

Classe Login:

La classe Login implementa la schermata di accesso all'applicazione. Permette all'utente di inserire username o email e password per effettuare il login, oppure di accedere alla schermata di registrazione per nuovi utenti. La classe si occupa di gestire l'interfaccia grafica (GUI) e di inoltrare le richieste al Controller secondo il pattern MVC (Model-View-Controller).

Attributi principali:

- **JTextField login:** campo di testo per l'inserimento di username e password.
- **JButton buttonInvio:** pulsante per confermare il login.
- **JPanel mainPanel:** pannello principale che contiene tutta l'interfaccia della schermata di login
- **JPanel textPanel, loginPanel:** pannelli secondari per organizzare la grafica
- **JPasswordField passwordField1:** campo per l'inserimento della password con caratteri nascosti
- **JButton buttonRegistrati:** pulsante per accedere alla schermata di registrazione
- **Controller controller:** riferimento al controller che gestisce la logica di login e registrazione

Metodi principali:

- **Login():** costruttore della classe login, qui si inizializza la GUI, si settano i placeholder, si configurano i listener per gli eventi e si crea il controller che gestirà le operazioni di login e registrazione.
- **actionPerformed(ActionEvent e):** gestisce gli eventi dei pulsanti, invia i dati di login al controller o apre la schermata di registrazione
- getter e setter per i componenti principali, utili per interagire con l'interfaccia da altre classi o dal controller

Classe DashBoardAdmin:

Classe GUI che rappresenta la dashboard dell'amministratore. Permette l'interazione con le funzionalità di gestione dei voli, bagagli e smarrimenti, come per la dashboard utente, le azioni vengono inoltrate al controller per mantenere la separazione tra logica e interfaccia.

Attributi principali:

- **JPanel dashboardAdminPage:** pannello principale che contiene tutta la dashboard dell'amministratore.
- **JLabel adminLabel:**etichetta per mostrare informazioni sull'admin, come l'username.
- **JPanel homePage, welcomePanel, welcomeText:** pannelli per organizzare i con- tenuti principali e il messaggio di benvenuto.
- **JLabel welcomeTextLabel:**etichetta che mostra il messaggio di benvenuto personalizzato con l'ID dell'admin.
- **JButton logoutUser:** pulsante per effettuare il logout.
- **JComboBox comboBox1:** menu a tendina con le azioni selezionabili dall'amministratore.
- **JLabel outputLabel:** etichetta usata per mostrare messaggi di feedback o output dopo azioni amministrative.
- **JDialog choiceDialog:** dialog modale per finestre di conferma o scelte particolari.
- **String username:** username o ID dell'amministratore, usato per personalizzare l'interfaccia.
- **Controller controller:** riferimento al controller per delegare la logica delle azioni.

Metodi principali:

- **DashBoardAdmin(String username, Controller controller):** costruttore della dashboard per amministratore. Inizializza la GUI, imposta le azioni disponibili nella combo box, mostra l'identità dell'amministratore e attiva i listener per la gestione eventi.
- **getDashBoardAdminPage():** restituisce il pannello principale della dashboard per l'inserimento nel frame
- getter e setter per i componenti principali, utili per interagire con l'interfaccia da altre classi o dal controller

4 Package DAO e Implementazione

Il package DAO (Data Access Object) si occupa della comunicazione tra l'applicazione java e il database relazionale (PostgreSQL), l'obiettivo di questo package è garantire un'interfaccia stabile e coerente per accedere ai dati.

4.1 Interfacce DAO

Le interfacce DAO definiscono in modo astratto i metodi da implementare per il corretto funzionamento dell'applicazione e della gestione dei dati.

Interfaccia UtenteDAO:

L'interfaccia UtenteDAO contiene i metodi comuni a tutti i tipi di utente del sistema indipendentemente dal ruolo specifico:

- visualizzaVoli()
- cercaBagaglio(Prenotazione prenotazione, Utentegenerico utente)
- cercaBagaglio(Bagaglio bagaglio, Utentegenerico utente)
- cercaVolo(Volo v)

Interfaccia UtenteGenericoDAO:

L'interfaccia UtenteGenericoDAO estende l'interfaccia UtenteDAO e contiene i metodi specifici per le operazioni che possono essere eseguite dagli utenti generici del sistema:

- listaPrenotazioni(UtenteGenerico utente)
- prenotaVolo(UtenteGenerico ug, Volo volo, Passeggero p, Arraylist<Bagaglio> ab)
- modificaPrenotazione(Prenotazione prenotazione, Arraylist<Bagaglio> ab)
- segnalaSmarrimento(Bagaglio bagaglio, UtenteGenerico u)
- cercaPrenotazione(UtenteGenerico utente, String nome, String cognome)
- cercaPrenotazione(UtenteGenerico utente, int numeroBiglietto)

poichè estende l'interfaccia UtenteDAO include anche i metodi visualizzaVoli(), cercaBagaglio(Prenotazione prenotazione, Utentegenerico utente), cercaBagaglio(Bagaglio bagaglio, Utentegenerico utente), cercaVolo(Volo v).

Interfaccia AmministratoreDAO:

Questa interfaccia estende UtenteDAO e include i metodi specifici per le operazioni amministrative del sistema aeroportuale:

- inserisciVolo(Volo volo)
- aggiornaVolo(Volo volo)
- modificaGate(VoloInPartenza volo)
- aggiornaBagaglio(Bagaglio bagaglio, StatoBagaglio stato)
- visualizzaSmarrimento()
- cercaPasseggero(Passeggero passeggero)

poichè estende l'interfaccia UtenteDAO include anche i metodi visualizzaVoli(), cercaBagaglio(Prenotazione prenotazione, Utentegenerico utente), cercaBagaglio(Bagaglio bagaglio, Utentegenerico utente), cercaVolo(Volo v).

Interfaccia LoginDAO:

Questa interfaccia include i metodi per la gestione dell'autenticazione e registrazione degli utenti:

- login(String email, String password)
- registrazione (String email, String password)

4.2 ImplementazionePostgresDAO

Il package implementazionePostgresDAO contiene le implementazioni concrete delle interfacce DAO, specifiche per il database PostgreSQL, ogni classe implementa i metodi definiti nelle relative interfacce.

Classi e metodi:

Classe UtenteGenericoImplementazionePostgresDAO

Questa classe implementa l'interfaccia UtenteGenericoDAO e consente a un utente generico di interagire con il sistema di gestione voli tramite PostgreSQL.

Metodi Principali:

- **UtenteGenericoImplementazionePostgresDAO():** è il costruttore e inizializza la connessione al database.
- **visualizzaVoli():** recupera l'elenco completo dei voli disponibili nel sistema.
- **cercaBagaglio(Prenotazione prenotazione, Utentegenerico utente):** cerca i bagagli associati ad una specifica prenotazione. Restituisce una lista con il bagaglio se trovato.
- **cercaBagaglio(Bagaglio bagaglio, Utentegenerico utente):** Cerca un bagaglio specifico in base all'ID. Restituisce una lista con il bagaglio se trovato.
- **cercaVolo(Volo v):** cerca un volo in base a criteri specifici.
- **listaPrenotazioni(UtenteGenerico utente):** Recupera tutte le prenotazioni effettuate da un utente specifico.
- **prenotaVolo(UtenteGenerico ug, Volo volo, Passeggero p, Arraylist<Bagaglio> ab):** esegue la prenotazione di un volo da parte di un utente autenticato.
- **modificaPrenotazione(Prenotazione prenotazione, Arraylist<Bagaglio> ab):** aggiorna l'elenco dei bagagli associati a una prenotazione.
- **segnalaSmarrimento(Bagaglio bagaglio, UtenteGenerico u):** permette all'utente di segnalare come smarrito un bagaglio a suo nome.
- **cercaPrenotazione(UtenteGenerico utente, String nome, String cognome):** cerca tutte le prenotazioni effettuate da un utente in cui il passeggero corrisponde al nome e cognome forniti.
- **cercaPrenotazione(UtenteGenerico utente, int numeroBiglietto):** cerca una prenotazione specifica tramite numero di biglietto.
- **generapostoRandom:** genera in modo pseudo-casuale un numero di posto tra 1 e 100 che verrà assegnato a una prenotazione nel momento della conferma.

Inoltre al suo interno si trova la classe statica **CustomExc** che estende RuntimeException. Il suo scopo è lanciare eccezioni personalizzate per gestire errori specifici dell'implementazione. Viene lanciata quando si verificano problemi durante le operazioni di accesso al database.

Classe AmministratoreImplementazionePostgresDAO

Questa classe implementa l'interfaccia AmministratoreDAO fornendo tutte le funzionalità amministrative del sistema aeroportuale, gestisce operazioni avanzate sui voli, gate, bagagli, e segnalazioni.

Metodi Principali:

- **AmministratoreImplementazionePostgresDAO():** è il costruttore della classe, inizializza la connessione al database.
- **visualizzaVoli():** recupera l'elenco completo dei voli disponibili nel sistema.
- **cercaBagaglio(Prenotazione prenotazione, Utentegenerico utente):** cerca i bagagli associati ad una specifica prenotazione. Restituisce una lista con il bagaglio se trovato.
- **cercaBagaglio(Bagaglio bagaglio, Utentegenerico utente):** Cerca un bagaglio specifico in base all'ID. Restituisce una lista con il bagaglio se trovato.
- **cercaVolo(Volo v):** cerca un volo in base a criteri specifici.
- **cercaPasseggero(Passeggero passeggero):** cerca e restituisce i dati e le prenotazioni di un passeggero in base al codice del documento.
- **inserisciVolo(Volo volo):** inserisce un nuovo volo nel database, specificando tutte le informazioni essenziali.
- **aggiornaVolo(Volo volo):** aggiorna i dati di un volo già esistente, è aggiornabile solo il campo della destinazione/origine.
- **modificaGate(VoloInPartenza volo):** modifica il gate di un volo in partenza.
- **aggiornaBagaglio(Bagaglio bagaglio, StatoBagaglio stato):** aggiorna lo stato di un bagaglio specifico nel database.
- **visualizzaSmarrimento():** recupera tutti i bagagli che risultano attualmente smarriti nel sistema.

Inoltre al suo interno si trova la classe statica **CustomExc** che estende RuntimeException. Il suo scopo è lanciare eccezioni personalizzate per gestire errori specifici dell'implementazione. Viene lanciata quando si verificano problemi durante le operazioni di accesso al database.

Classe LoginImplementazionePostgresDAO

Questa classe implementa l'interfaccia LoginDAO fornendo la logica concreta per l'autenticazione e la registrazione degli utenti. La classe gestisce la verifica delle credenziali e l'inserimento di nuovi utenti nel database.

Metodi Principali:

- **LoginImplementazionePostgresDAO():** è il costruttore della classe, inizializza la connessione al database.
- **login(String email, String password):** verifica le credenziali di accesso per un utente o un amministratore.
- **registrazione(String email, String password):** registra un nuovo utente nel database.

4.3 Connessione al Database

Per la connessione al database si fa uso della classe **ConnessioneDatabase**.

Questa classe implementa il pattern Singleton per la gestione centralizzata della connessione al database PostgreSQL. Fornisce un punto di accesso univoco per tutte le operazioni di database, garantendo efficienza delle risorse e controllo degli accessi.

Metodi Principali:

- **ConnessioneDatabase()**: costruttore privato che inizializza la connessione PostgreSQL con i parametri di configurazione. Gestisce le eccezioni di connessione e registra lo stato nel log.
- **getInstance():** metodo statico per ottenere l'istanza singleton della classe. Crea l'istanza se non esiste, altrimenti restituisce quella esistente.
- **getConnection():** metodo per ottenere la connessione al database, con controllo di validità e riconnessione automatica se necessario.

5 Package Controller

il Package controller tramite la classe **Controller** gestisce la logica applicativa dell'interfaccia grafica. Funziona come coordinatore tra la GUI e le DAO che interagiscono con il database PostgreSQL. Supporta sia le operazioni lato utente che lato amministratore.

Attributi principali:

- Amministratore admin
- UtenteGenerico utente

Metodi principali:

- **handlerVisualizzaPrenotazioni(DashBoardUser d):**

  Visualizza in una tabella tutte le prenotazioni effettuate dall'utente corrente.

- **mostraDettagliVolo(int codiceVolo):**

  In base al codiceVolo recupera e mostra tutti i dettagli di un determinato volo.

- **initContenitore(JScrollPane scrollPane):**

  Inizializza un pannello contenitore per il dialog delle prenotazioni.

- **handlerPrenotaVolo(JPanel panel, JDialog dialog, DashBoardUser d):**

  Costruisce il form per effettuare una nuova prenotazione e la salva nel database.

- **handlerCercaPrenotazione(JPanel panel, Jlabel label, JDialog dialog, DashBoardUser d):**

  Consente la ricerca di prenotazioni dell'utente tramite numero biglietto o dati anagrafici.

- **mostraPopupPrenotazioni(ArrayList<Prenotazione> risultati, JLabel label):**

  Mostra un popur con l'elenco delle prenotazioni trovate. Ogni riga include un bottone per modificarne il numero di bagagli.

- **creaRigaPrenotazione(Prenotazione p, JLabel label):**

  Crea una riga grafica contenente i dati sintetici di una prenotazione e un bottone di modifica.

- **initRiga():**

  Inizializza una riga con border e colore di sfondo specifici.

- **handlerCercaBagaglio(JPanel panel, JDialog dialog):**

  Permette la ricerca di un bagaglio associato alle proprie prenotazioni

- **sincronizzaCampi(JTextField campo1, JTextField campo2):**

  Sincronizza due campi testo in modo che solo uno dei due possa contenere testo alla volta. Quando l'utente scrive nel primo campo, il secondo viene svuotato automaticamente e viceversa.

- **initListaPanel(ArrayList<Bagaglio> lista):**

  Inizializza una lista di bagagli in un pannello con righe.

- **initScrollPanel(JPanel listaPanel):**

  Inizializza un pannello di scorrimento per visualizzare i risultati della ricerca bagagli.

- **mostraPopupRisultati(ArrayList<Bagaglio> lista):**

  Mostra i risultati di una ricerca bagagli per utente in una finestra modale

- **handlerSegnalaSmarrimento(JPanel panel, JDialog dialog, DashBoardUser d):**

  Consente all'utente autenticato di segnalare lo smarrimento di un proprio bagaglio.

- **handlerCercaVolo(JPanel panel, JDialog dialog):**

  Costruisce dinamicamente l'interfaccia per la ricerca di un volo, l'utente può scegliere se cercare un volo per codice oppure per origine e destinazione.

- **eseguiRicercaVolo(JTextField codiceField, JText Field origineField, JTextField destinazioneField, JDialog dialog):**

  Esegue la ricerca del volo in base ai dati inseriti dall'utente nel form.

- **mostraRisultatiNelPopup(ArrayList<Volo> voli, JDialog parentDialog):**

  Mostra in un popup modale tutti i voli trovati dalla ricerca, uno sotto l'altro.

- **costruisciDettagliVolo(Volo v):**

  Restituisce una stringa formattata con tutti i dettagli del volo, da usare per l'interfaccia testuale nel popup.

- **selectedItem(String item, DashBoardUser d):**

  Gestisce la selezione di un'azione dall'interfaccia utente. In base all'elemento selezionato, apre il dialog appropriato per l'azione richiesta.

- **handlerInserisciVolo(JPanel panel, JDialog dialog, DashBoardAdmin d):**

  Crea e inserisce un nuovo volo nel database tramite DAO, supporta voli in arrivo e in partenza con campi condizionati.

- **creaFormPanel():**

  Crea un pannello verticale a una colonna per moduli grafici.

- **creaCampoStandard(String placeholder):**

  Crea un campo di immissione con stile coerente all'interfaccia scura.

- **creaCheckBox():**

  Crea una checkBox "in Arrivo" con stile grafico coerente.

- **configuraCheckBox(JCheckBox check, JTextField localitaField, JTextField gateField):**

  Configura il comportamento dinamico della checkbox "In arrivo", modifica i campi località e gate in base alla selezione.

- **costruisciVoloDaInput(JTextField codiceField, JTextField compagniaField, JTextField dataField, JTextField orarioField, boolean inArrivo, JTextField localitaField, JTextField gateField):**

  Costruisce un oggetto Volo (in arrivo o in partenza) partendo dai valori inseriti in un form grafico.

- **aggiungiComponentiForm(JPanel form, JComponent... componenti):**

  Aggiunge in ordine tutti i componenti forniti al pannello specificato.

- **handlerAggiornaVolo(DashBoardAdmin d):**

  Mostra la lista dei voli e consente di selezionare un volo da modificare.

- **getRiga(DashBoardAdmin d, Volo v):**

  Crea una riga grafica per un volo, con etichetta e bottone di modifica.

- **getJButton(DashBoardAdmin d, Volo v):**

  Crea un bottone di modifica per un volo, che apre un popup per modificarne i dettagli

- **handlerModificaGate(JPanel panel, JDialog dialog, DashBoardAdmin d):**

  Permette la modifica del gate di un volo in partenza.

- **habdlerCercaBagaglioAdmin(JPanel panel, JDialog dialog, DashBoardAdmin d):**

  Ricerca bagagli tramite ID o numero prenotazione, con risultati visualizzati in tabella.

- **creaFormPanel2():**

  Crea un pannello verticale a una colonna con numero righe personalizzato.

- **eseguiRicercaBagagli(JTextField idField, JTextField prenotazioneField):**

  Esegue la ricerca di bagagli tramite ID o numero prenotazione. Lancia eccezioni se i campi sono entrambi vuoti o entrambi compilati.

- **mostraRisultatiBagagli(ArrayList<Bagaglio> risultati, JDialog dialog):**

  Mostra i risultati della ricerca bagagli in una finestra a scorrimento.

- **initFormPanelAB():**

  Inizializza un pannello per il form di modifica bagaglio.

- **handlerAggiornaBagaglio(JPanel panel, JDialog dialog, DashBoardAdmin d):**

  Consente la modifica dello stato di un bagaglio

- **handlerVisualizzaSmarrimenti(DashBoardAdmin d):**

  Mostra la lista dei bagagli smarriti segnalati dagli utenti.

- **handlerCercaPasseggero(JPanel panel, JDialog dialog):**

  Crea un modulo grafico per cercare i dati e le prenotazioni di un passeggero a partire dal suo ID documento.

- **eseguiRicercaPasseggero(JTextField idField, JDialog dialog):**

  Esegue la ricerca delle prenotazioni per un passeggero, utilizzando l'ID del documento fornito nel campo di input

- **mostraPrenotazioniInPopup(ArrayList<Prenotazione> prenotazioni, JDialog parentDialog):**

  Mostra un popup modale con i dati del passeggero e tutte le sue prenotazioni trovate

- **selectedItem(String item, DashBoardAdmin d):**

  Metodo che gestisce la selezione di un elemento dal menu a tendina della dashboard admin, in base all'elemento selezionato, apre un dialog con le opzioni corrispondenti.

- **creaCampo(String placeholder):**

  Crea un JTextField con stile grafico predefinito e colore coerente.

- **creaBottoneConAzione(String testo, Runnable azione, JDialog dialog):**

  Crea un bottone con testo e azione associata, dopo l'esecuzione dell'azione, chiude automaticamente il dialog specificato.

- **interfacciaRegistrazione():**

  MOstra l'interfaccia grafica per la registrazione di un nuovo utente, effettua un check di esistenza e salva su database se valido.

- **getRegistrati(JTextField emailField, JPasswordField passwordField, JDialog dialog):**

  Crea un bottone di registrazione con azione associata, controlla i campi e registra l'utente se valido.

- **registraUtente(String email, String password):**

  Registra un nuovo utente in PostgreSQL.

- **tableBuild():**

  Costruisce l'HTML di base per la tabella dei voli.

- **visualizzaVoli(DashBoardUser view):**

  Carica e visualizza i voli in una tabella HTML nella dashboard utente.

- **visualizzaVoli(DashBoardAdmin view):**

  Carica e visualizza i voli in una tabella HTML nella dashboard admin.

- **mostraPopupModificaVolo(Volo volo):**

  Mostra una dialog modale per la modifica dei dati di un volo esistente, consente di aggiornare il ritardo, stato, origine/destinazione e gate (solo per voli in partenza).

- **creaEtichetta(String testo):**

  Crea una JLabel con il testo fornito, utilizzando uno stile coerente con l'interfaccia scura dell'applicazione.

- **login(String email, String password):**

  Esegue l'autenticazione verificando credenziali tramite DAO, imposta internamente code utente o code admin se l'accesso ha successo.

- **mostraPopupModificaPrenotazione(Prenotazione p):**

  Mostra una dialog per modificare il numero di bagagli associati a una prenotazione.

6 Conclusioni

La realizzazione del progetto è stata un'importante occasione per mettere in pratica le competenze acquisite durante il percorso formativo. Grazie al lavoro di squadra e al continuo confronto tra colleghi, è stato possibile affrontare con successo le sfide proposte dalla traccia assegnata, raggiungendo gli obiettivi prefissati e realizzando un'applicazione funzionante, robusta e conforme ai requisiti iniziali.

In particolare, ci siamo impegnati con grande attenzione a rendere il progetto il più possibile coerente con la traccia fornita, assicurando che ogni funzionalità fosse correttamente implementata e adeguatamente documentata, rispettando pienamente i vincoli e le indicazioni iniziali.

L'esperienza ci ha permesso di consolidare non solo le nostre abilità tecniche, ma anche quelle organizzative e comunicative, indispensabili per la gestione efficace di un progetto condiviso. Rimangono aperti interessanti margini di miglioramento e ulteriori sviluppi futuri, che potrebbero arricchire ulteriormente il prodotto ottenuto.

Si ringraziano infine i docenti per il supporto e i preziosi suggerimenti forniti durante lo sviluppo del progetto.