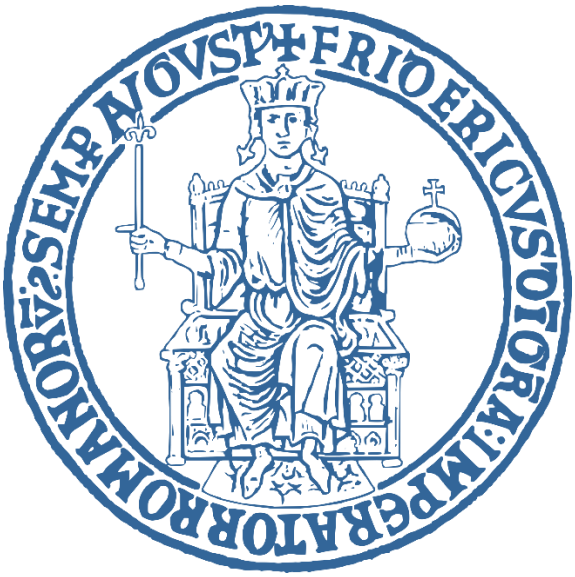


DIPARTIMENTO DI INFORMATICA

Corso di Programmazione Object-Oriented

Anno Accademico 2024/2025

Progettazione e sviluppo di un APPLICATIVO JAVA

Docenti: Porfirio Tramontana, Bernardo Breve

Autori: Antonio Andrea Montella (N86005652), Adriano Montella (N86005823)

Sommario

1. PANORAMICA DEL PROGETTO

L’applicativo è stato creato utilizzando **IntelliJ**, in linguaggio **Java**. Implementa un sistema completo per la **gestione di voli, prenotazioni e biglietti aeroportuali**. Il progetto utilizza un'architettura a tre livelli (**3-tier**), con interfaccia grafica realizzata in **Swing** e database realizzato con **PostgreSQL**.

1.1. VALUTAZIONE DEL PATTERN PROGETTUALE

Nella progettazione, avremmo potuto adottare un modello che prevedesse che le classi del "model" contenessero tutta la logica di business relativa alla propria classe. Così facendo, il controller avrebbe delegato tutto alle classi del model, rendendo il progetto più "compatto". In quest'architettura i dati sarebbero mantenuti esclusivamente dalle classi del package "model", in modo temporaneo. Questo vuol dire che al lancio dell'applicazione non ci sarebbero dati (previe istanze preesistenti nel codice, nello specifico nel Controller), e alla terminazione tutti i dati letti verrebbero quindi cancellati!

Per questo motivo abbiamo bisogno di una soluzione che ci permetta di mantenere parte dei dati. Questo ci ha portato alla soluzione architetturale composta **BCE + DAO**, che ci ha permesso di integrare al progetto un **DBMS**, ovvero un Sistema di Gestione di Database. Nel nostro specifico caso, abbiamo aggiunto all’applicativo la libreria **postgresql-42.7.5.jar**, tramite il pom.xml fornitoci a lezione. Il pattern BCE + DAO permette quindi al nostro applicativo di ottenere una **separazione delle responsabilità** tra i vari livelli:

- **Boundary (confine)**: è il livello di interazione con l'utente o altri programmi. Nel nostro caso corrisponde al package "GUI";
- **Control (controllo)**: è il livello di coordinamento, che fornisce i vari servizi al livello “Boundary” e gestisce tutto il resto. Qui vengono implementati tutti gli algoritmi di logica (anche detti di business) di funzionamento del programma. Nel nostro caso, corrisponde al package "controller", contenente solo una (grande) classe Controller.java;
- **Entity (entità)**: è il livello del modello, che tiene traccia di tutti gli elementi appartenenti al dominio del problema risultati dall'analisi. Nel nostro caso, corrisponde al package "model", contenente tutte le classi i cui oggetti sono istanziati e chiamati dal Controller.

1.2. STRUTTURA DEI PACKAGE

- **controller**: logica di controllo;
- **dao**: interfacce Data Access Object;
- **database**: connessione e configurazione DB;
- **GUI**: interfacce grafiche Swing;
- **implementazioniPostgresDAO**: implementazioni concrete DAO;
- **model**: modello del dominio.

2. INTERFACCIA UTENTE

- **Sistema di Autenticazione:**
  - LandingPageLogin: pagina di login principale;
  - Register: form di registrazione nuovo utente.
- **Dashboard Utente:**
  - UserDashboard: dashboard principale utente standard;
  - AreaPrivata: area personale con prenotazioni;
  - OpzioniRicerca: ricerca e prenotazione voli;
  - VoliUser: visualizzazione voli disponibili.
- **Dashboard Amministratore:**
  - AdminDashboard: dashboard principale amministratore;
  - AreaPrivataAdmin: area personale amministratore;
  - VoliAdmin: gestione completa voli.
- **Dialog:**
  - TicketDialog: visualizzazione dettagli ticket;
  - ModificaPrenotazioneDialog: modifica prenotazioni esistenti;
  - DialogInserisciVolo: inserimento nuovo volo;
  - DialogModificaVolo: modifica voli esistenti;
  - DialogDatiPasseggero: inserimento dati passeggero.

3. FUNZIONALITÀ PRINCIPALI

Per utenti standard:

- registrazione e autenticazione;
- ricerca voli per tratta e data;
- prenotazione voli (max 9 passeggeri);
- gestione prenotazioni personali;
- visualizzazione e modifica ticket;
- cancellazione prenotazioni.

Per amministratori:

- tutte le funzionalità utente standard;
- inserimento e modifica voli;
- gestione stati voli e ritardi;
- visualizzazione tutte le prenotazioni;
- gestione completa del sistema.

4. VERIFICA DELLA QUALITÀ DEL CODICE

È stata effettuata anche un'analisi approfondita della qualità del codice tramite il plugin "**SonarQube**". Il codice è stato **pulito da tutti gli errori ritenuti gravi**, e sono stati smaltiti la maggior parte degli errori ritenuti "di basso impatto" dal plugin.

Sono da notare con maggiore attenzione le segnalazioni del plugin relative al codice generato automaticamente da Swing UI. È stato scelto di non cambiare il codice generato in automatico dal plugin predisposto alle interfacce grafiche, come d'altronde consigliato dai commenti stessi di Swing UI. Pertanto, sono stati ignorati tutti i warning relativi a quei frammenti di codice!

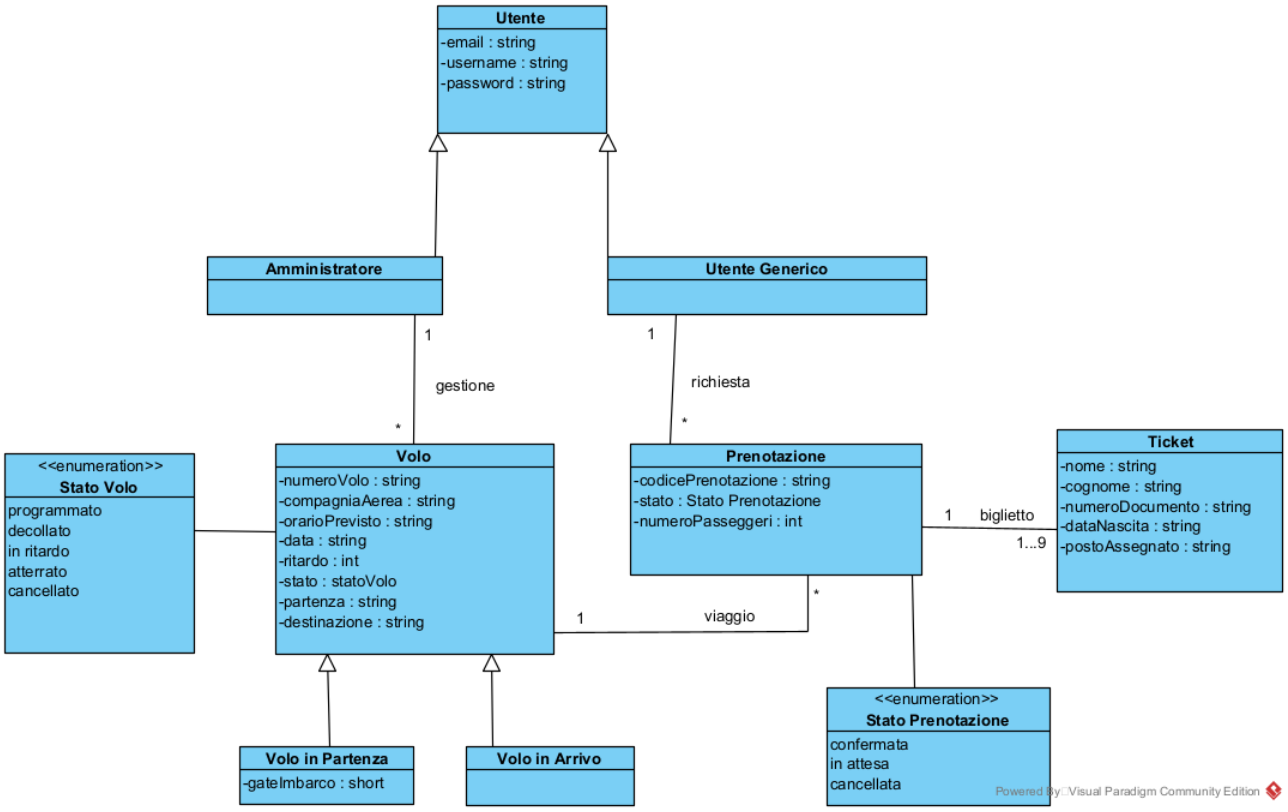
5. TECNOLOGIE UTILIZZATE

Maggiori dettagli sulle tecnologie adottate sono disponibili nel file README.md consultabile al link del repository. Inoltre, è possibile trovare anche un riassunto su come utilizzare l'applicativo realizzato!

- **Java 24**: linguaggio di programmazione;
- **Swing**: framework per interfaccia grafica;
- **PostgreSQL**: database relazionale;
- **JDBC**: connettività database;
- **Maven**: build automation e gestione dipendenze;
- **IntelliJ IDEA**: IDE di sviluppo.

6. DOMINIO DEL PROBLEMA

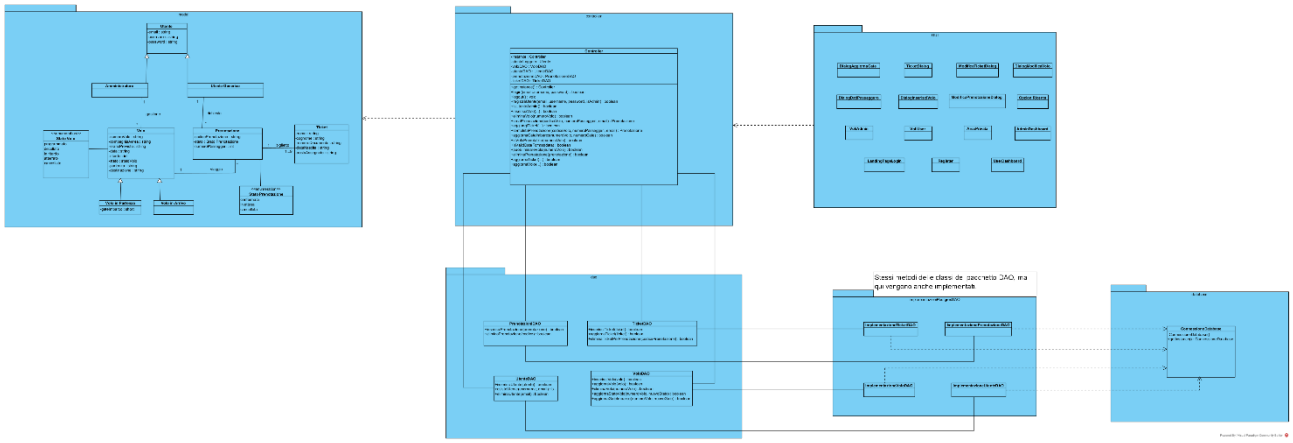
6.1 CLASS DIAGRAM



6.2 RELAZIONI TRA LE ENTITÀ ED EREDITARIETÀ

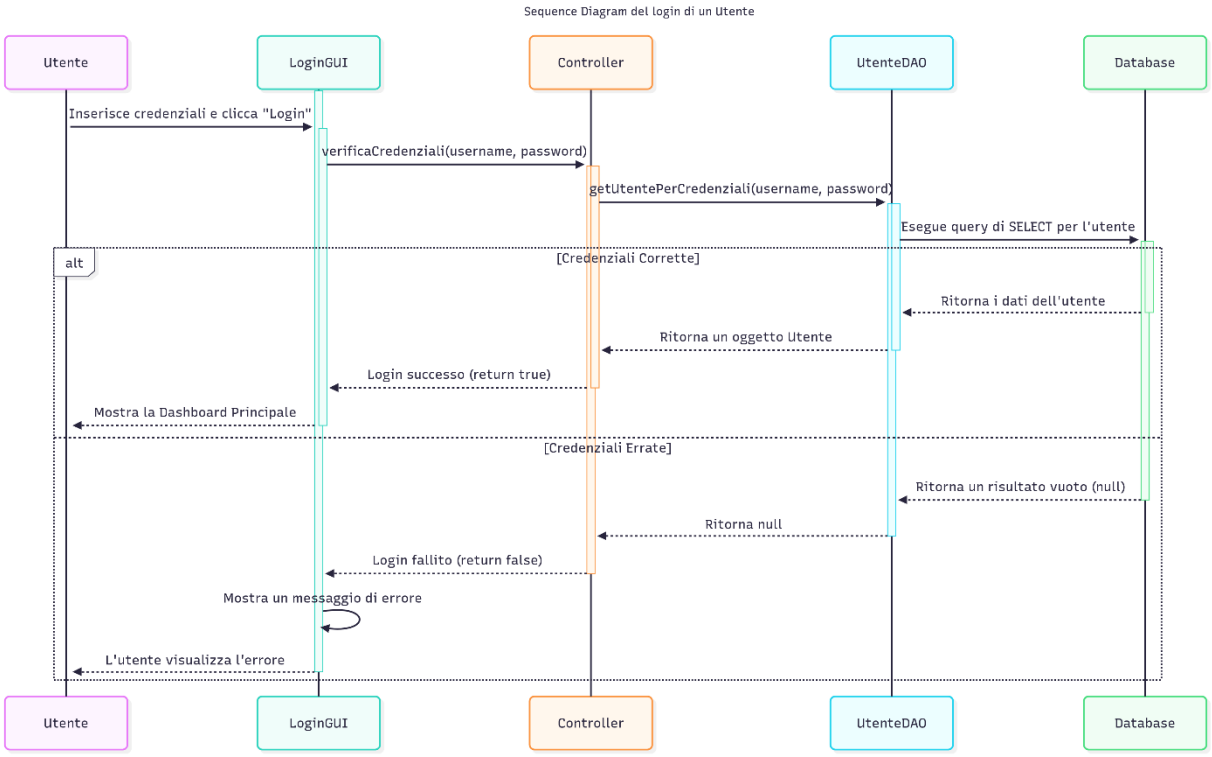
- **Utente Generico** ↔ **Prenotazione** (1:N): un utente può richiedere più prenotazioni;
- **Amministratore** ↔ **Volo** (1:N): un amministratore può gestire più voli;
- **Volo** ↔ **Prenotazione** (1:N): un volo può avere più prenotazioni;
- **Prenotazione** ↔ **Ticket** (1:1…9): una prenotazione può contenere un massimo di 9 ticket;
- **Utente Generico** e **Amministratore** sono sottoclassi della classe **Utente**;
- **Volo in Partenza** e **Volo in Arrivo** sono sottoclassi della classe **Volo**.

6.3 BCE + DAO DIAGRAM

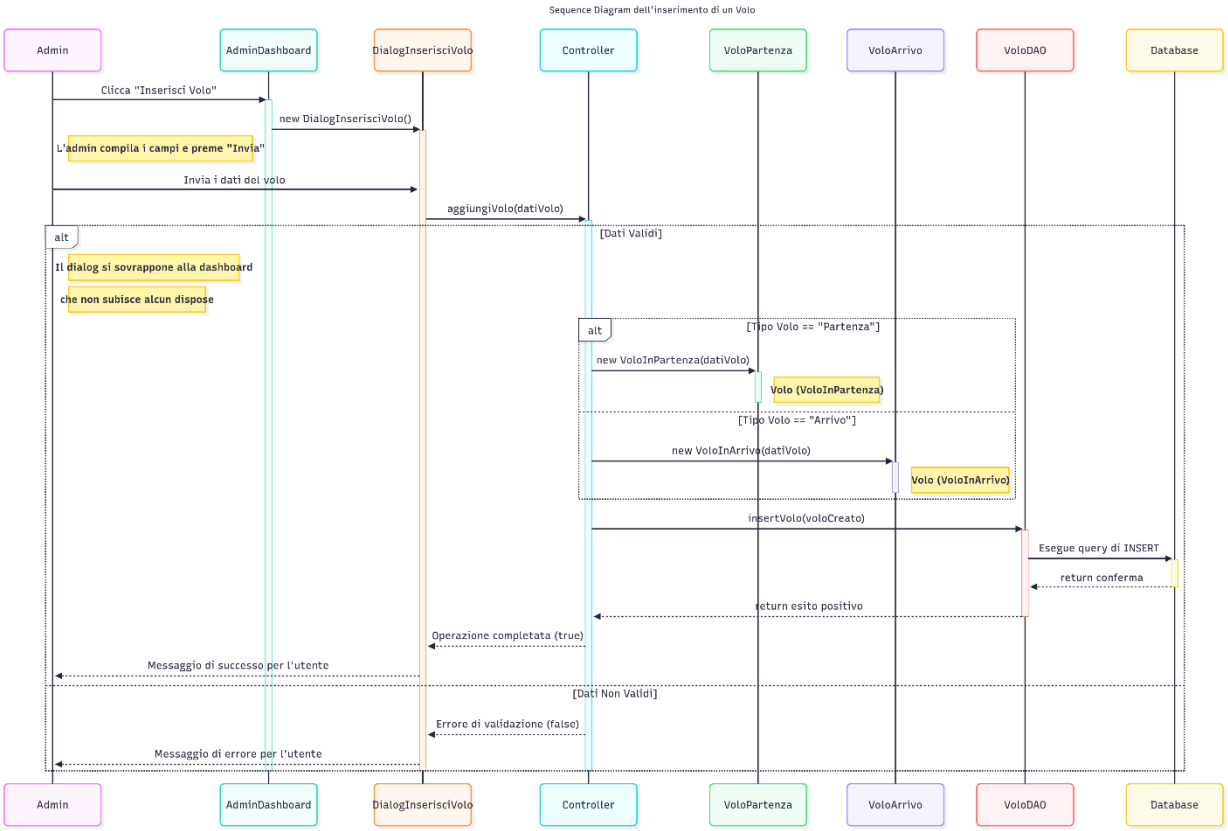


7.0 SEQUENCE DIAGRAMS

7.1 LOGIN UTENTE



7.2 INSERIMENTO NUOVO VOLO



8. DETTAGLI GENERICI

Regole di Business:

- **Prenotazioni**: massimo 9 passeggeri per prenotazione;
- **Voli**: codice alfanumerico univoco;
- **Ritardi**: espressi in minuti;
- **Posti**: codifica alfanumerica;
- **Utenti**: email e username univoci nel sistema.

Pattern Singleton: È stato anche implementato il pattern Singleton, che prevede un Controller unico per tutta l'applicazione, per non crearne più volte uno nuovo.

Javadoc: Il plugin "Javadoc" in è stato utilizzato al fine di generare automaticamente la documentazione in formato Javadoc a partire dai commenti nel codice Java. Funziona analizzando le dichiarazioni di classi, metodi e campi, e produce file HTML descrittivi. Aiuta a mantenere documentazione coerente e facilmente consultabile direttamente dall'IDE.