Aeroporto Napoli

Studenti:
Michele Lo Franco N86005184
Lorenzo Massaro N86005027
Claudia Lattarulo N86005348

1 Descrizione del progetto

Il presente progetto si concentra sulla realizzazione di un sistema software avanzato e integrato per la prenotazione e la gestione operativa dei voli all'interno di un moderno terminale aeroportuale. L'obiettivo primario è affrontare le complessità delle operazioni aeree e aeroportuali attraverso una soluzione digitale che garantisca efficienza, accuratezza e una migliore esperienza sia per il personale operativo che per i passeggeri.

L'obiettivo fondamentale del sistema è **semplificare, standardizzare e digitalizzare** tutti i processi operativi critici relativi alla movimentazione dei passeggeri, alla gestione dei velivoli e al coordinamento dei servizi di terra. La sua implementazione è finalizzata a:

- **Miglioramento della Velocità Operativa:** Ridurre i tempi di attesa e di elaborazione dei dati, velocizzando le procedure di check-in, imbarco e controllo voli, grazie a un'interfaccia utente (UI) studiata per essere intuitiva.
- **Gestione Dati in Tempo Reale:** Fornire una visione unificata e aggiornata in tempo reale su tutti gli aspetti operativi, inclusi lo stato dei voli (partenze/arrivi), l'assegnazione dei gate, e le informazioni sui passeggeri.

Il sistema software è progettato per offrire un set completo di funzionalità, indirizzando le esigenze sia del *back-office* che delle interfacce con l'utenza:

1. **Modulo di Prenotazione e Biglietteria:**
    - Ricerca e selezione voli basata su parametri multipli (destinazione, data).
    - Gestione completa delle prenotazioni, incluse modifiche.
2. **Gestione Dati Operativi:**
    - **Passeggeri:** Archiviazione e gestione dei profili dei passeggeri.
    - **Voli e Orari:** Mantenimento di un database aggiornato degli itinerari, degli orari di arrivo/partenza previsti ed effettivi, e monitoraggio degli eventuali ritardi o cancellazioni.
    - **Gate e Bagagli:** Assegnazione dinamica e monitoraggio dello stato di occupazione dei gate di imbarco/sbarco e dei nastri bagagli.
3. **Visualizzazione in Tempo Reale:**
    - Dashboard operativi per il personale aeroportuale con lo stato aggiornato di tutti i voli.
    - Interfacce utente (FIDS - Flight Information Display System) per i passeggeri con informazioni chiare e tempestive su partenze, arrivi e assegnazione gate.

L'applicazione è sviluppata adottando un'architettura **modulare**. Questo approccio non solo facilita lo sviluppo e la manutenzione, ma assicura anche la flessibilità necessaria per l'adattamento futuro:

- **Modularità:** La suddivisione in moduli distinti (e.g., Modulo Prenotazioni, Modulo Gestione Gate, Modulo Reportistica) consente l'aggiornamento o la sostituzione di componenti specifici senza impattare l'intero sistema.
- **Estendibilità:** La progettazione aperta permette la facile integrazione con altri sistemi aeroportuali esistenti (ad esempio, sistemi di gestione bagagli, sistemi di sicurezza, sistemi di controllo del traffico aereo) e l'aggiunta di futuri moduli di espansione, come l'introduzione di servizi self-service avanzati per i passeggeri (e.g., check-in mobile, tracciamento bagagli).

2 Requisiti

Il sistema sviluppato ha come obiettivo la gestione completa del processo di prenotazione e amministrazione dei voli all'interno di un terminal aeroportuale.

Sono stati individuati due principali tipi di utenti, **l'utente standard** e **l'amministratore**, ognuno con specifici permessi e funzionalità.

2.1 Requisiti Utente

L'utente ha la possibilità di interagire con il sistema per consultare e gestire le proprie prenotazioni in modo semplice e intuitivo.
In particolare, può:

- **Cercare voli disponibili**, filtrando per destinazione, data o fascia oraria.
- **Effettuare una prenotazione** selezionando il volo desiderato, il numero di passeggeri e le eventuali opzioni aggiuntive.
- **Visualizzare le prenotazioni effettuate**, con tutti i dettagli relativi a volo, orari, passeggeri e numero di prenotazione.
- **Gestire i propri bagagli**, visualizzando le informazioni relative ad esso.

Queste funzionalità sono pensate per garantire all'utente un'esperienza fluida, sicura e trasparente nella gestione delle operazioni di viaggio.

2.2 Requisiti Amministratore

L'amministratore dispone di privilegi avanzati per la gestione del sistema e dei dati.
In particolare, può:

- **Aggiungere nuovi voli** inserendo informazioni come codice del volo, destinazione, data, orario, compagnia e disponibilità di posti.
- **Modificare i voli esistenti**, aggiornando i dati in caso di variazioni operative (ritardi, cancellazioni, cambio di gate, ecc.).
- **Cancellare voli**, ad esempio in caso di problemi tecnici o di pianificazione.
- **Gestire le prenotazioni**, potendo creare, aggiornare o eliminarle su richiesta dell'utente o per motivi amministrativi.
- **Gestire i bagagli**, verificando il numero e le caratteristiche dei bagagli associati a ciascuna prenotazione.

L'interfaccia riservata all'amministratore è progettata per garantire un controllo completo e centralizzato del sistema, mantenendo un elevato livello di sicurezza e coerenza dei dati.

2.3 Requisiti non funzionali

- **Usabilità:** l'interfaccia deve risultare chiara e facilmente navigabile.
- **Affidabilità:** il sistema deve gestire correttamente eventuali errori o eccezioni senza perdita di dati.
- **Scalabilità:** l'architettura deve consentire l'aggiunta futura di nuove funzionalità (es. check-in online o acquisto del biglietto online)
- **Sicurezza:** l'accesso alle funzioni amministrative deve essere protetto da autenticazione.
    -

3 Approccio alla realizzazione

Il primo passo per la creazione del progetto è stata la creazione del Diagramma.
Durante il primo incontro ci siamo occupati di analizzare la traccia, capire i punti fondamentali e sviluppare un piano di azione.
Dopo la creazione del diagramma, abbiamo deciso di suddividere la task e distribuire i compiti.

Successivamente abbiamo creato la repository su github.
Ci siamo occupati di capire il funzionamento di Git e le sue istruzioni.
Effettuati diversi test per controllare se tutto fosse stato settato correttamente.

Arrivati a questo punto, abbiamo iniziato con la scrittura del codice.
Iniziando dalle classi, ci siamo suddivisi poi varie parti del codice:
ogni sera durante questo periodo ci sono stati degli incontri per discutere brevemente dei progressi fatti in giornata e “connettere” tutti i nostri codici per far in modo che tutto funzionasse correttamente.

Una volta pronto lo scheletro del progetto, abbiamo iniziato a lavorare alla documentazione e al database.
In particolare, quest’ultimo era necessario per il corretto funzionamento del software, ed è stato strutturato senza troppi intoppi grazie al class model fatto nelle fasi iniziali.

Finito anche quello, abbiamo potuto lavorare su dettagli minori, lavorando inizialmente sulla grafica, che è stata decisa di lasciare pulita per permettere un uso semplice e immediato del software.

In fase finale abbiamo effettuato i test, scovato errori e bug, e man mano pensavamo anche a risolvere il problema in questione, arrivando cosi alla conclusione del software.

Il primo passo nella creazione del progetto è consistito nella realizzazione del Diagramma.

Durante l'incontro iniziale, abbiamo analizzato la traccia, identificato i punti cruciali e sviluppato un piano d'azione.

A seguito della creazione del diagramma, si è proceduto alla suddivisione delle attività e alla distribuzione dei compiti.

Successivamente, è stata creata la repository su GitHub.

Ci siamo dedicati alla comprensione del funzionamento di Git e delle sue istruzioni.

Sono stati eseguiti diversi test per verificare il corretto settaggio di tutti gli elementi.

A questo punto, si è dato inizio alla fase di scrittura del codice.

Partendo dalla definizione delle classi, ci siamo suddivisi le varie sezioni del codice:

in questo periodo, ogni sera si sono tenuti degli incontri per discutere i progressi giornalieri e "integrare" i codici individuali, assicurando il corretto funzionamento complessivo.

Una volta completata la struttura portante del progetto, si è iniziato a lavorare alla documentazione e al database.

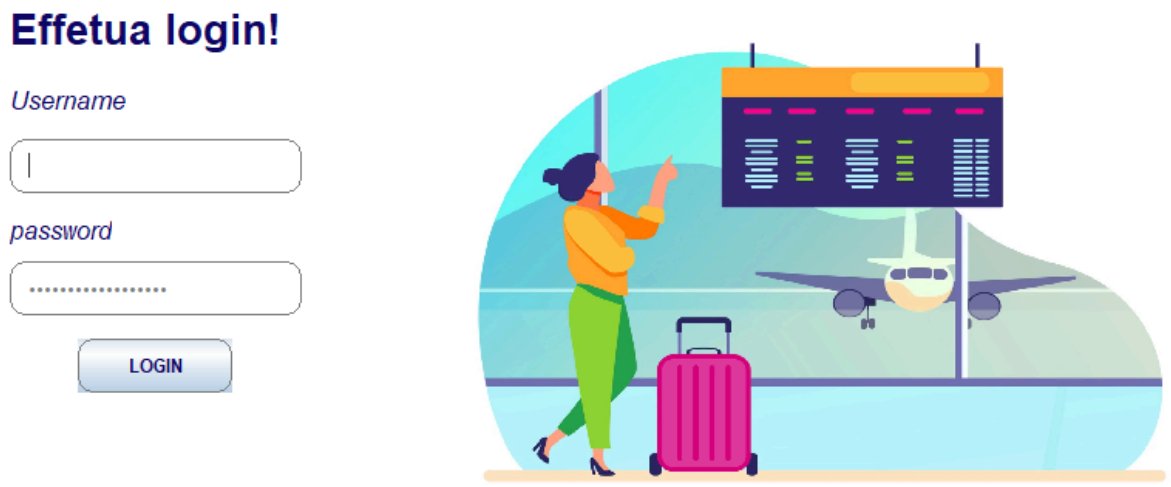
In particolare, il database si è rivelato indispensabile per il corretto funzionamento del software, ed è stato strutturato senza particolari difficoltà grazie al *class model* elaborato nelle fasi preliminari.

Terminata anche questa fase, abbiamo potuto concentrarci sui dettagli minori, lavorando inizialmente sull'interfaccia grafica, che è stata concepita con un design pulito per facilitare un utilizzo semplice e immediato del software.

Nella fase finale, sono stati eseguiti i test, sono stati identificati e corretti errori e *bug*, portando così alla conclusione dello sviluppo del software.

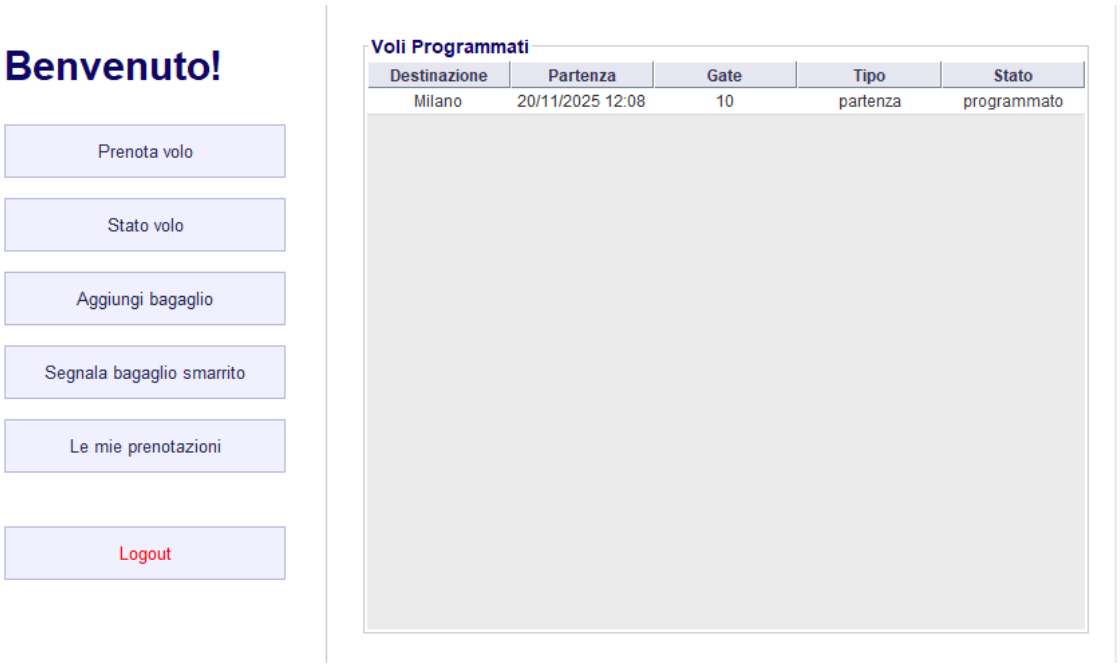
4 Manuale di Istruzioni

Aprendo la prima volta il software, si presenta una schermata di login



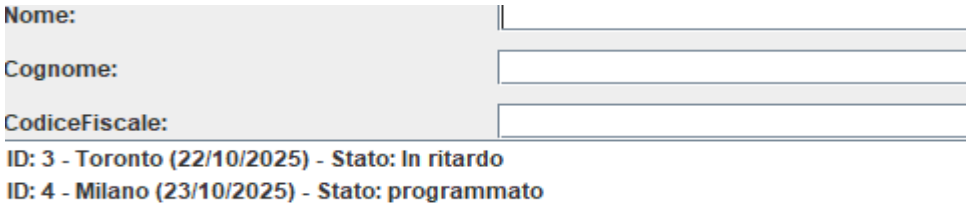
dove un utente (o amministratore) può accedere al software utilizzando le proprie credenziali personali.

4.1 LATO UTENTE



L'utente, all'interno del software, ha a disposizione tutte le opzioni per l'organizzazione del proprio volo personale.

Con **Prenota Volo**, può prenotare (da una lista voli aggiornata da un amministratore) il proprio viaggio.



.
Dovrà inserire i suoi dati personali, scegliere il volo desiderato e risulterà prenotato in quel volo.

Con **stato volo**, l'utente può verificare lo stato dei voli prenotati che ha prenotato.

ID: 2 - Torino (16/10/2025) - Stato: programmato
ID: 3 - Toronto (22/10/2025) - Stato: In ritardo
ID: 1 - Miami (16/10/2025) - Stato: programmato
ID: 4 - Milano (23/10/2025) - Stato: programmato

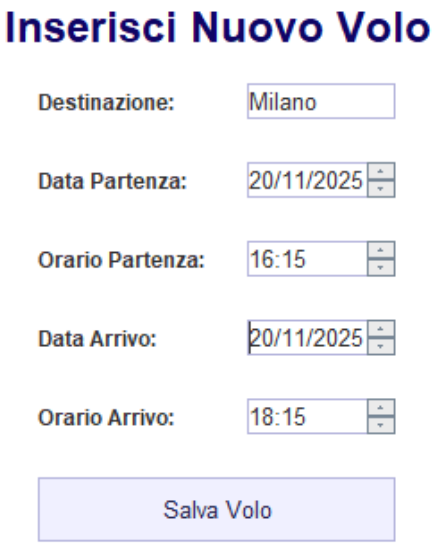
Con **aggiungi bagaglio**, l'utente seleziona il volo desiderato, e inserendo il codice del suo bagaglio può prenotarlo per il volo selezionato.

Infine, con **segnala bagaglio smarrito**, l'utente può segnalare all'amministratore lo smarrimento del proprio bagaglio.
Proprio come per "**aggiungi bagaglio**", anche in questo caso sarà necessario selezionare il volo di riferimento e il codice bagaglio perso.

4.2 LATO AMMINISTRATORE

Dopo aver fatto l'accesso con le credenziali da amministratore, avremmo molti più strumenti a disposizione per la gestione dei voli.

Partendo da **inserisci volo in partenza**, l'amministratore può inserire un nuovo volo aggiungendo la destinazione e gli orari. (N.B. L'aeroporto di partenza sarà sempre quello di Napoli)



Ugualmente, con **inserisci volo in arrivo**, si può inserire un volo programmato ad atterrare l'aeroporto di Napoli (N.B. L'aeroporto di arrivo sarà sempre quello di Napoli)

Con il bottone **Assegna gate** un amministratore deve selezionare il gate di riferimento per ogni volo.

Il bottone **Visualizza smarrimenti** permette all'amministratore di visualizzare i reclami di smarrimento effettuati dagli utenti nella "**sezione bagaglio smarrito**" e le relative informazioni.

Con il bottone **Modifica volo** l'amministratore può visualizzare e modificare lo stato di un volo, segnalando se quest'ultimo dovesse essere in orario, in ritardo o cancellato.

Con il bottone "**Aggiorna bagaglio**" l'amministratore può avere una panoramica di tutti i bagagli di un volo selezionato, inoltre può modificare lo stato di quest'ultimo nel caso di necessità.

Con il bottone **Visualizza passeggero** l'amministratore, sempre selezionando un volo specifico, può vedere l'intera lista degli utenti prenotati per quel volo.

Tutti gli strumenti sopraelencati permettono all'amministratore di avere una panoramica completa dei voli e tutto ciò che ruota intorno a essi.

5. DOCUMENTAZIONE DIAGRAMMA UML

UTENTE: In questa entità abbiamo inserito due attributi *Login* e *Password* per accedere al sistema informativo dell'aeroporto, e il metodo *VisualizzaVoli()* che serve per visualizzare i voli in partenza da Napoli. L'utente si suddivide in UTENTE GENERICO e AMMINISTRATORE. Abbiamo preferito utilizzare la relazione di aggregazione tra utente generico e amministratore con utente perché queste due entità sono a loro volta utenti.

AMMINISTRATORE: Anche qui come utente generico abbiamo inserito gli stessi attributi ma con metodi diversi siccome l'amministratore ha compiti diversi: *inserisceVolo() modificaVolo() aggiornaBagaglio() visualizza bagaglio().*

UTENTE GENERICO: In questa entità abbiamo inserito gli attributi *Nome, Cognome* e *Codice_Fiscale* per identifiicarsi e per la prenotazione del volo infatti, come metodi abbiamo inserito *prenotaVolo() cercaPrenotazione() segnalaBagaglio().* L'utente generico è collegato a prenotazione con la relazione *effettua* (1,*).

PRENOTAZIONE: Gli attributi sono *Numero_Bagaglio, Posto_Assegnato, Stato*. Come metodo fondamentale abbiamo inserito il *checkin().* La prenotazione ha diversi stati che abbiamo inserito in *StatoDellaPrenotazione* dove all'interno ci sono gli attributi: *Conferma, InAttesa, Cancellata.* La Prenotazione è collegata con la relazione *possiede* (1, *) a BAGAGLIO, ed è anche collegata all'entità VOLO con la relazione *riguarda* (1,*). Si collega a sua volta a PASSEGGERO con la relazione *esegue* (1,1).

PASSEGGERO: Gli attributi sono le caratteristiche per riconoscerlo: *Nome, Cognome, NumeroDocumento.*

BAGAGLIO: Contiene le informazioni riferite ad esso come il *Codice* e lo *Stato.*

VOLO: Ha le informazioni riferite ad esso come il Codice, Compagnia_aerei, Data_Volo, Destinazione, Partenza, StatoVolo. Il volo ha diversi stati che abbiamo inserito in *Stato_Volo: InRitardo, Atterrato Decollato, Cancellato.*

VOLO_PARTENZA_ARRIVO: E' una dipendenza di VOLO quindi abbiamo preferito inserire la relazione di dipendenza. Questa entità ha i seguenti attributi: *Gate, AereoportoPartenza=Napoli, AereoportoArrivo=Napoli*. VOLO_PARTENZA_ARRIVO è collegata con la relazione *boarding* (1, *) a GATE.

GATE: Abbiamo inserito la condizione necessaria come attributo cioè il *numero* che lo identifica.

Class Diagram.