
UNIVERSITÀ
DEGLI STUDI
DI UDINE

# Documentazione finale di Air Connect

**Corso: Cybersecurity**

**Nome Studente: Stefano Panico**

**Matricola: 169091**

## Indice generale

## Introduzione

**Air Connect** è una piattaforma web progettata per semplificare e ottimizzare l'intero processo di gestione dei **voli**, rendendo l'esperienza dell'utente **intuitiva** e **sicura**.

Il sistema offre funzionalità avanzate per gli **utenti finali**, consentendo loro di effettuare il **check-in**, **acquistare** e **cancellare biglietti** con pochi passaggi.

Grazie a una solida architettura **backend** e a un'interfaccia **frontend** user-friendly, gli utenti possono accedere facilmente ai servizi di **prenotazione voli.**

In aggiunta, **Air Connect** include un modulo di **gestione** dedicato agli **amministratori**, che ha il compito di monitorare e gestire le operazioni quotidiane legate ai **voli**. Gli **amministratori** possono **creare**, **modificare** e **cancellare voli**, assicurando che l'offerta di itinerari sia sempre aggiornata e allineata alle esigenze degli utenti. Il sistema offre inoltre la possibilità di **assegnare ruoli** di **amministratore** ad altri utenti, permettendo una gestione **flessibile** e distribuita delle risorse.

L'**autenticazione** degli utenti e l'accesso alle funzionalità amministrative sono gestiti tramite un sistema di **controllo dei ruoli**, che garantisce che solo gli utenti **autorizzati** possano eseguire operazioni sensibili, come la gestione dei **voli** e dei **permessi**. Inoltre, il sistema è **scalabile** e può facilmente adattarsi a futuri miglioramenti, come l'integrazione con altri sistemi di **viaggio** o l'aggiunta di nuove funzionalità per gli **utenti finali.**

## Obiettivi del progetto

L'obiettivo principale di questo progetto è sviluppare un'applicazione web sicura e affidabile, in grado di proteggere i dati degli utenti da potenziali minacce informatiche.

Sono state implementate diverse misure di sicurezza per garantire la riservatezza, l'integrità e la disponibilità delle informazioni.

In particolare, il progetto si propone di raggiungere i seguenti obiettivi di sicurezza:

- **Implementazione di un database:** per l'archiviazione delle informazioni degli utenti.
- **Gestione delle sessioni utente**: per prevenire accessi non autorizzati e furti di identità.
- **Implementazione del protocollo HTTPS**: per garantire la cifratura dei dati durante la trasmissione tra il browser dell'utente e il server.
- **Gestione dei cookie**: utilizzo di cookie sicuri e conformi alle normative sulla privacy per l'autenticazione e la gestione delle preferenze degli utenti.
- **Password criptate**: implementazione di un sistema di criptazione delle password, al fine di non memorizzare le password in chiaro.
- **Prevenzione di attacchi comuni**: Implementazione di misure di sicurezza per prevenire attacchi comuni, come cross-site request forgery (CSRF).

Attraverso il raggiungimento di questi obiettivi, il progetto mira a creare un ambiente online sicuro e affidabile per gli utenti

## Architettura di Air Connect

Il progetto *Air Connect* è strutturato secondo un'**architettura a microservizi**, suddividendo il sistema in componenti indipendenti che comunicano tra loro tramite API REST. Questo approccio migliora la scalabilità, la manutenzione e l'affidabilità dell'applicazione.

### 1. Frontend (React.js)

Il frontend è sviluppato in **React.js** e si occupa di gestire l'interfaccia utente, interagendo con il backend per ottenere e inviare dati.

La struttura del frontend è suddivisa nei seguenti moduli:

- `assets/` → Contiene risorse multimediali (immagini, icone, ecc.).
- `components/` → Racchiude componenti riutilizzabili, come il layout principale (`Layout.js`).
- `pages/` → Contiene le pagine principali dell'applicazione, tra cui:
  - `HomePage.js` (pagina principale)
  - `LoginPage.js` e `RegisterPage.js` (gestione autenticazione)
  - `FlightsPage.js` (visualizzazione voli disponibili)
  - `AddFlightPage.js` e `ManagementFlightPage.js` (gestione amministrativa dei voli)
  - `HistoryPage.js` (storico delle prenotazioni)
  - `ProfilePage.js` (profilo utente)
  - `UserManagementPage.js` (gestione utenti con ruoli di admin)
- `services/` → Gestisce la comunicazione con il backend attraverso API REST. Ogni file si occupa di una specifica funzionalità:
  - `authService.js` (autenticazione)
  - `flightService.js` e `flightAdminService.js` (gestione voli)
  - `ticketService.js` (gestione biglietti)
  - `historyService.js` (storico operazioni)
  - `userService.js` (gestione utenti e ruoli)
- `App.js` → Punto di ingresso principale del frontend, che gestisce il routing e l'inizializzazione dell'applicazione.

## 2. Backend (Node.js + Express)

Il backend è sviluppato in **Node.js** con il framework **Express** e gestisce la logica di business dell'applicazione. Si interfaccia con un database per gestire utenti, voli, biglietti e storico delle operazioni.

La struttura del backend è suddivisa nei seguenti moduli:

- `config/` → Contiene la configurazione del database (`index.js`).
- `middleware/` → Contiene il middleware per la gestione dei permessi utente, ovvero `authMiddleware.js`.
- `models/` → Definisce la struttura delle tabelle nel database:
  - `flightModel.js` (dati relativi ai voli)
  - `historyModel.js` (storico operazioni)
  - `ticketModel.js` (gestione biglietti)
  - `userModel.js` (gestione utenti e ruoli)
- `routes/` → Contiene le API REST che gestiscono le operazioni principali:
  - `authRoute.js` (autenticazione e gestione login/register)
  - `flightRoute.js` (visualizzazione voli per gli utenti e gestione voli per admin)
  - `historyRoute.js` (storico prenotazioni e azioni)
  - `ticketRoute.js` (gestione acquisto, cancellazione e check-in dei biglietti)
  - `userRoleRoute.js` (gestione ruoli amministrativi)
- `app.js` → Avvia il server Express e configura tutte le rotte e i middleware.

## 3. Comunicazione tra Frontend e Backend

Il frontend comunica con il backend tramite **API RESTful**, utilizzando richieste HTTPS per inviare e ricevere dati in formato **JSON**. Le chiamate sono le seguenti:

**authRoute.js**

- **POST** `/api/auth/register` - Registrazione di un nuovo utente.
- **POST** `/api/auth/login` - Login di un utente.
- **GET** `/api/auth/profile` - Visualizzazione del profilo utente.
- **PUT** `/api/auth/profile/edit` - Modifica del profilo utente.
- **POST** `/api/auth/logout` - Logout dell'utente.

**ticketRoute.js**

- **POST** `/api/ticket/purchase` - Acquisto di un biglietto.
- **POST** `/api/ticket/cancel/:ticketId` - Cancellazione di un biglietto.
- **POST** `/api/ticket/checkin/:ticketId` - Check-in di un biglietto.

**flightRoute.js**

- **POST** `/api/flight/flights` - Aggiunta di un nuovo volo (solo per admin).
- **GET** `/api/flight/flights/:flightNumber` - Lettura di un volo specifico.
- **PUT** `/api/flight/flights/:id` - Aggiornamento di un volo (solo per admin).
- **DELETE** `/api/flight/flights/:id` - Cancellazione di un volo (solo per admin).
- **GET** `/api/flight/search` - Ricerca dei voli con validazione, paginazione e filtri.
- **GET** `/api/flight/flights` - Lettura di tutti i voli con paginazione e filtri.

**historyRoute.js**

- **GET** `/api/history/read` - Lettura dello storico delle operazioni dell'utente autenticato.

**userRoleRoute.js**

- **GET** `/api/user/users` - Recupero di tutti gli utenti.
- **PUT** `/api/user/users/:id/role` - Aggiornamento del ruolo di un utente.

## 4. Sicurezza e gestione dei ruoli

L’applicazione include un sistema di **autenticazione e autorizzazione** basato su **JWT (JSON Web Token)** per proteggere le API. Gli utenti sono classificati in due ruoli:

- **Utente normale (User)**: può prenotare e cancellare biglietti, visualizzare lo storico.
- **Admin**: può gestire voli e assegnare ruoli.

L'accesso a determinate API è regolato tramite **middleware di autorizzazione** (`authMiddleware.js`).