Air Connect -
Presentazione del Progetto

UNIVERSITÀ
DEGLI STUDI
DI UDINE

Esame: Cybersecurity

Studente: Stefano Panico

Matricola: 169091

Indice presentazione

1. Introduzione

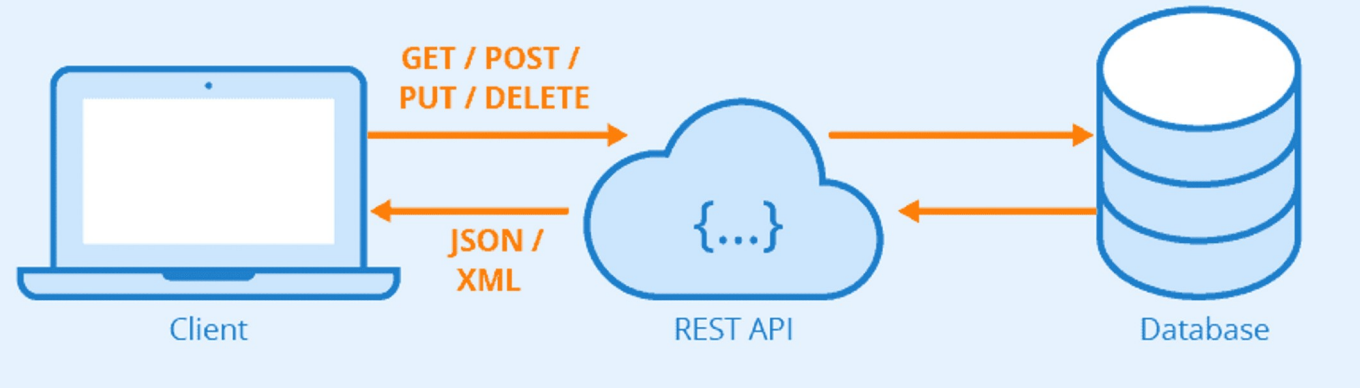
- Air Connect è una **piattaforma web** ideata per semplificare la gestione delle **prenotazioni aeree**.
- Grazie a un'interfaccia user-friendly, agli utenti è consentito:
  - Acquistare biglietti
  - Cancellare biglietti
  - Completare il check-in
  - Modificare informazioni del proprio profilo
  - Guardare lo storico dei biglietti
- Il sistema è pensato anche per gli **amministratori**, che possono svolgere i seguenti compiti:
  - Creare voli
  - Modificare voli
  - Cancellare voli
  - Modificare ruoli di altri utenti

2. Obiettivo

- L'obiettivo principale di Air Connect è offrire una **piattaforma digitale sicura ed efficiente** per la gestione delle prenotazioni aeree.
- Gli aspetti chiave del progetto sono:
  - Utilizzo di un **database**
  - Gestione della **sessione** utente
  - Utilizzo del protocollo **HTTPS**
  - Gestione dei **Cookie**
  - Utilizzo di **password criptate**

3. Architettura

- Air Connect è basato su un'architettura a **microservizi**. Questo approccio permette di separare le varie funzionalità in **servizi indipendenti**.
- Tale architettura si può suddividere in due macro-componenti:
  - **Frontend**: sviluppato in **React.js**, riguarda tutto ciò che è l'interfaccia del sistema
  - **Backend:** realizzato con **Node.js** ed **Express**, gestisce la logica di business e la sicurezza
- L'interazione tra frontend e backend avviene tramite **API REST**.



4. Sicurezza e Gestione Ruoli (1)

4.1. Database

- Database utilizzato: **SQLite**
- Gestione: **ORM Sequelize** (Node.js)
- Tabelle principali:
  - **User** → Memorizza utenti con email, password (hash), nome, cognome e ruolo (user/admin)
  - **Flight** → Contiene i dettagli dei voli: numero, orari, destinazione e posti disponibili
  - **Ticket** → Registra i biglietti acquistati, associati ai voli, con stato e check-in.
  - **History** → Tiene traccia delle operazioni degli utenti (acquisto, cancellazione, check-in) con timestamp
- Vantaggi:
  - **Integrazione diretta** con Node.js
  - **Struttura modulare**
  - **Facilità di migrazione** ad altri DBMS in futuro

4. Sicurezza e Gestione Ruoli (2)

4.2. Sessione utente

- Tecnologie utilizzate:
  - **Express-session** (Node.js)
  - **JSON Web Token** (JWT)
  - **Middleware di autenticazione**
- L'utente effettua il **login** e riceve un **JWT**. Il **token** viene inviato con ogni richiesta **API** per l'autenticazione. Il backend verifica il token e concede l'accesso alle **risorse autorizzate**.
- Vantaggi:
  - **Sicurezza** → Nessuna gestione di sessioni lato server
  - **Scalabilità** → JWT è stateless, ideale per architetture a microservizi
  - **Efficienza** → Le richieste API non necessitano di controlli continui nel database

4. Sicurezza e Gestione Ruoli (3)

4.3. Protocollo HTTPS

- Tecnologie utilizzate:
  - **Protocollo HTTPS** (HyperText Transfer Protocol Secure)
  - **Certificati SSL/TLS**
- Il server utilizza un certificato **SSL/TLS** per criptare la comunicazione. Tutti i dati inviati tra client e server vengono protetti dalla **crittografia**. Il browser verifica il certificato per garantire una **connessione sicura**.
- Vantaggi:
  - **Protezione dati** → Impedisce attacchi Man-in-the-Middle (MITM)
  - **Integrità** → Garantisce che i dati non siano modificati durante il trasferimento

4. Sicurezza e Gestione Ruoli (4)

4.4. Cookie

- Tecnologie utilizzate:
  - **Express-session** (Node.js)
  - **Cookie** con opzioni **HTTPOnly** e **SameSite**
- I cookie vengono utilizzati per mantenere la **sessione attiva**. Il flag **HttpOnly** impedisce l'accesso ai cookies da parte di script lato client (prevenendo **attacchi XSS**). L'opzione **SameSite** protegge dai **Cross-Site Request Forgery** (CSRF).
- Vantaggi:
  - **Protezione contro XSS** → Impedisce ad uno script JavaScript malevolo di accedere ai cookie
  - **Mitigazione CSRF** → Evita richieste non autorizzate da siti terzi

4. Sicurezza e Gestione Ruoli (5)

4.5. Password criptate

- Tecnologie utilizzate:
  - **bcrypt.js** per l'hashing
  - **Salt** per aumentare la sicurezza delle password
- L'utente inserisce una **password** al momento della registrazione. La password viene **criptata** con **bcrypt** e un **salt casuale**. Durante il login, la password inserita viene confrontata con la versione criptata della stessa salvata in **database**.
- Vantaggi:
  - **Protezione dei dati** → Le password salvate non sono in chiaro
  - **Protezione da attacchi brute-force**
  - **Conformità agli standard di sicurezza** → Adeguato a normative come GDPR e ISO 27001

email	password	name	surname
user1@example.com	$2a$10$WNiWerCzNYNUJJzff4WNn.ZcLu7vZE0QJ5l.VdYoHJF.jDglxpddW	Roberto	Roberti
user2@example.com	$2a$10$8II3FINGmZBAjZboBI3z4OFZTN3C1qmY6wP5SBEWEN6mkAmESmvKq	Anna	Gialli
user3@example.com	$2a$10$iCPrM3oBnqFI6EuWfPJ9fuw55N3Ze0zkMbP7gzvRe64DJ5G90KWDi	Marco	Ferri

5. Struttura delle Directory Backend (Node.js)

6. Struttura delle Directory Frontend (React.js)

frontend

|--- pages → **Pagine principali dell'applicazione**

| |--- HomePage.js → **Pagina principale**

| |--- LoginPage.js → **Gestione autenticazione**

| |--- RegisterPage.js → **Registrazione nuovi utenti**

| |--- FlightsPage.js → **Visualizzazione voli disponibili**

| |--- AddFlightPage.js → **Creazione voli (admin)**

| |--- ManagementFlightPage.js → **Gestione voli (admin)**

| |--- HistoryPage.js → **Storico delle prenotazioni**

| |--- ProfilePage.js → **Profilo utente**

| |--- UserManagementPage.js → **Gestione ruoli (admin)**

|--- services → **Comunicazione con API REST del Backend**

| |--- authService.js → **API Gestione utenti**

| |--- flightService.js → **API Visualizzazioni voli**

| |--- flightAdminService.js → **API per amministratori**

| |--- ticketService.js → **API Acquisto e gestione biglietti**

| |--- historyService.js → **API Storico operazioni utente**

|--- App.js → **Punto di ingresso principale dell'applicazione**

7. Comunicazione tra Frontend e Backend

- Tecnologie utilizzate:
  - **Axios** per l'invio di richieste HTTPS dal frontend
  - **Express.js** per creare API REST nel backend
  - **Formato JSON** per lo scambio di dati tra client e server
- Il **frontend** invia richieste HTTPS al backend utilizzando **Axios**. I Il **backend** (**Express.js**) elabora la richiesta e restituisce una risposta in **JSON**. Il frontend aggiorna l'interfaccia utente in base ai dati ricevuti.
- Vantaggi di Axios:
  - **Semplicità d'uso** → API chiare per richieste GET, POST, PUT, DELETE
  - **Supporto alle credenziali** → Opzione **withCredentials** per autenticazione sicura
  - **Gestione errori** → Possibilità di intercettare errori
- Endpoint principali:
  - **/api/auth** → Login, registrazione, gestione profilo
  - **/api/flight** → Ricerca e gestione voli
  - **/api/ticket** → Acquisto, cancellazione, check-in biglietti
  - **/api/history** → Storico transazioni

8. Come si utilizza Air Connect

Avviamo l’applicazione e accediamo tramite il seguente link:

https://localhost:8081

9. Conclusioni

- Air Connect vuole semplificare **l'intero processo** di gestione di voli aerei e biglietti sia per gli utenti, sia per gli amministratori.
- L'uso di un'architettura a **microservizi** garantisce modularità e manutenibilità.
- Le **misure di sicurezza avanzate** proteggono i dati degli utenti e garantiscono un accesso controllato.
- Possibili sviluppi futuri:
  - Integrazione con **sistemi di pagamento online**.
  - **Notifiche push** per aggiornamenti sui voli.
  - **Espansione del sistema** per supportare più compagnie aeree.