

# SkyTravel

*Report del progetto*

*Francesco Giovanni Pasqual*

Matricola: 901476

20 settembre 2025

Questo documento è stato scritto con L<sup>A</sup>T<sub>E</sub>X

# Indice

<b>1 Descrizione generale del sistema</b>	<b>3</b>
1.1 Flusso di alto livello . . . . .	3
<b>2 Modello dei dati</b>	<b>5</b>
2.1 Progettazione concettuale con modello orientato agli oggetti . . . . .	5
2.2 Progettazione logica con modello relazionale . . . . .	5
2.3 Entità principali (estratte dal codice): . . . . .	6
<b>3 API REST</b>	<b>7</b>
3.1 Autenticazione (/auth) . . . . .	7
3.2 Aeroporti (/aeroporti) . . . . .	8
3.3 Ricerca soluzioni (/soluzioni) . . . . .	8
3.4 Prenotazione posti (/booking) . . . . .	9
3.5 Checkout e pagamenti (/checkout) . . . . .	10
3.6 Area passeggero (/passeggero) . . . . .	10
3.7 Area compagnia (/compagnia) . . . . .	12
3.8 Area amministratore (/admin) . . . . .	13
<b>4 Autenticazione e gestione sessioni</b>	<b>15</b>
4.1 Flusso client . . . . .	15
4.2 Guardie di routing (Angular) . . . . .	15
4.3 HttpInterceptor: Authorization e refresh . . . . .	15
4.4 Middleware server: requireAuth e requireRole . . . . .	17
4.5 Tabella sessions come whitelist dei refresh . . . . .	17
4.6 Rotazione segreti (ROTATE_SECRETS_ON_START) . . . . .	17
4.7 Sicurezza: scelte e mitigazioni . . . . .	18
<b>5 Front end Angular</b>	<b>19</b>
5.1 Struttura . . . . .	19
<b>6 Screenshot e flussi per ruolo</b>	<b>20</b>
6.1 Passeggero . . . . .	20
6.2 Admin e compagnia aerea . . . . .	25
<b>7 Conclusioni</b>	<b>29</b>

# 1 Descrizione generale del sistema

SkyTravel TS è un'applicazione web per la ricerca di voli, la gestione delle compagnie aeree e l'acquisto di biglietti. Il sistema è composto da:

- Front end Angular (client) che gestisce la web app come una SPA (Single Page Application): interfaccia costruita con componenti composti da file HTML (struttura), CSS (stile) e TypeScript (comportamento). Alcuni sono inclusi direttamente nei template (*component nesting*), altri sono caricati dal Router dentro `<router-outlet>` in base al **path** dell'URL (definito in `app.routes.ts` con relative guardie). Styling con Tailwind CSS (framework CSS utility-first) e daisyUI (plugin/libreria di componenti per Tailwind) per usare componenti predefiniti e ridurre al minimo il CSS scritto a mano. Include **routing**, **guardie**, **HttpInterceptor** per allegare il token e gestire il refresh, e servizi per interagire con le API dichiarate nel server.
- Back end Node.js/Express (server) che si collega al database ed **espone le rotte REST** che gestiscono tutto lo scambio di dati tra client e server rendendo la web app dinamica. Comprende **autenticazione** e **autorizzazione**, **API** che leggono/scrivono dati dal DB, **upload e salvataggio di immagini** e **l'integrazione con Stripe** per la creazione e gestione dei pagamenti.
- Database relazionale (PostgreSQL) con script di inizializzazione in `db/init/`: **memorizza** e **gestisce** gli utenti e le sessioni; garantisce **integrità referenziale**, supporta **query e join** efficienti, **transazioni** per operazioni atomiche e vincoli per dati coerenti.
- Integrazione Docker che avvia **client, server e database in container separati** collegati su una rete interna: ogni componente è isolato ma comunica con gli altri per scambiarsi dati. Favorisce **portabilità** e **coerenza dell'ambiente** tra computer, facilita la **riproducibilità** delle build, l'**isolamento delle dipendenze**, il **versioning** dei servizi e un semplice **scaling** o sostituzione dei componenti.

L'architettura segue un pattern client-server con API REST stateless per la maggior parte delle operazioni. L'autenticazione usa JWT (access token nel client, refresh token HttpOnly in cookie) con gestione sessioni lato DB per revoca/validazione dei refresh token.

## 1.1 Flusso di alto livello

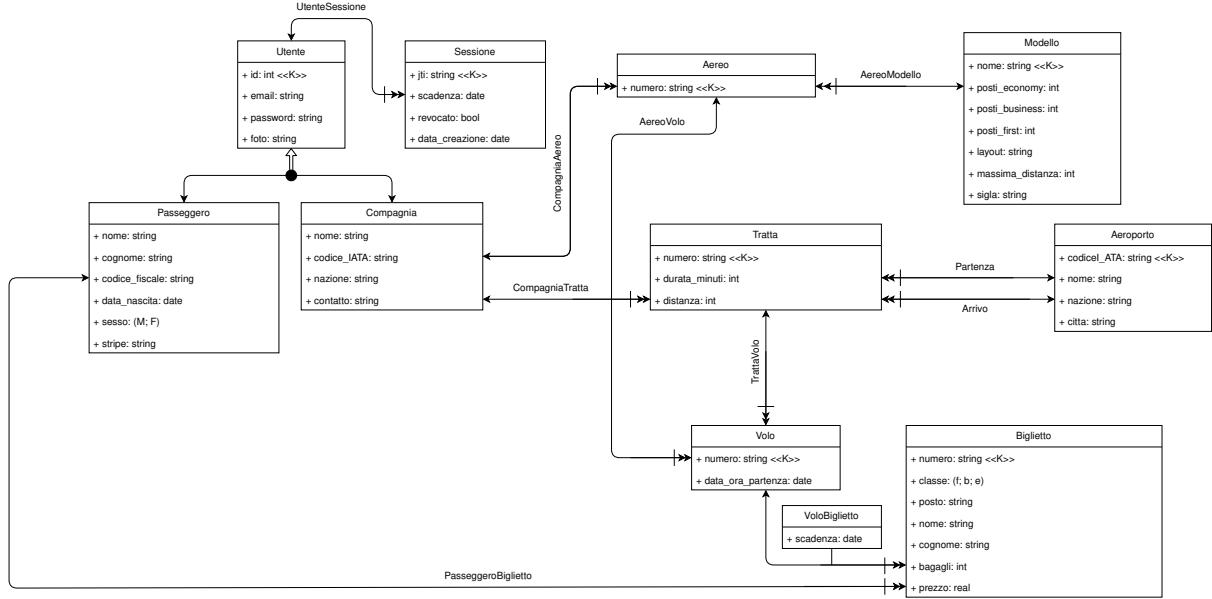
1. L'utente visita il front end Angular e, se necessario, effettua il login o registrazione per ottenere i ruoli: PASSEGGERO, COMPAGNIA o ADMIN (in base all'intervallo di ID utente).
2. Il client usa i servizi per chiamare le API REST. L'**HttpInterceptor** allega il bearer token e gestisce il refresh automatico quando l'access token scade. In questo modo è possibile gestire le richieste e le interazioni dal lato client.
3. Le rotte protette richiedono `requireAuth` e, ove necessario, `requireRole` per garantire l'accesso in base al ruolo.
4. Per quanto riguarda l'utente di tipo passeggero, le operazioni principali che può svolgere sono la prenotazione, la visualizzazione dei voli e del profilo, nonché la gestione dei biglietti. In particolare, ciò prevede: la ricerca degli itinerari, la selezione dei posti con trattenuta temporanea e la conferma dei biglietti dopo il pagamento al checkout. L'utente di tipo compagnia aerea, invece, può accedere al proprio profilo per consultare statistiche rilevanti e utilizzare gli appositi visualizzatori di

aerei, tratte e voli, ciascuno dei quali offre funzionalità dedicate alla loro gestione. Per quanto riguarda l'utente di tipo admin, questo può gestire gli utenti di tipo compagnia aerea o passeggero, eliminandoli oppure, nel caso delle compagnie aeree, aggiungendole sotto forma di invito. Le compagnie invitate vengono visualizzate nel pannello delle compagnie aeree in attesa, da cui è possibile cancellare l'invito. Sono inoltre presenti visualizzatori con refresh manuale per i passeggeri e per le compagnie aeree.

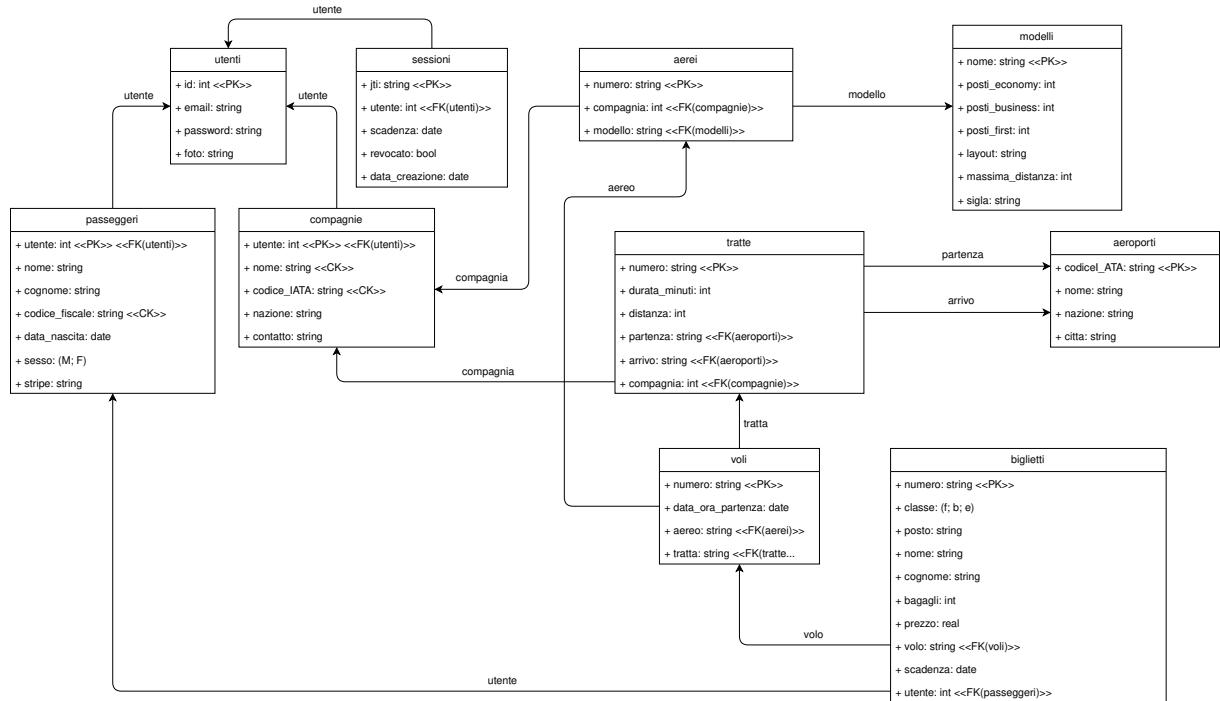
## 2 Modello dei dati

Il progetto utilizza PostgreSQL. Gli script di definizione e popolamento sono in db/init/\*.sql.

## 2.1 Progettazione concettuale con modello orientato agli oggetti



## 2.2 Progettazione logica con modello relazionale



### 2.3 Entità principali (estratte dal codice):

- **utenti** (id, email, password, foto)
- **sessioni** (jti, utente, scadenza, revocato, data creazione)
- **passeggeri** (utente, nome, cognome, codice fiscale, data nascita, sesso, stripe)
- **compagnie** (utente, nome, codice iata, nazione, contatto)
- **modelli** (nome, posti economy, posti business, posti first, layout, massima distanza, sigla)
- **aerei** (numero, compagnia, modello)
- **aeroporti** (codice iata, nome, nazione, citta)
- **tratte** (numero, durata minuti, distanza, partenza, arrivo, compagnia)
- **voli** (numero, data ora partenza, aereo, tratta)
- **biglietti** (numero, classe, posto, nome, cognome, bagagli, prezzo, volo, scadenza, utente)

### 3 API REST

Tutte le API sono esposte sotto il prefisso `/api`. Per le rotte protette è richiesto l'header `Authorization: Bearer <token>` (access token). Dove indicato, è necessaria anche la guardia di ruolo con i ruoli ammessi.

#### 3.1 Autenticazione (`/auth`)

**GET /auth/email** Pubblica. query: `email`. Scopo: verificare disponibilità email.

**POST /auth/register** Pubblica. body: `{ email, password, dati:{ nome, cognome, codiceFiscale, dataNascita, sesso } }`. Scopo: registrazione passeggero; crea utente e profilo passeggero.

**POST /auth/login** Pubblica. body: `{ email, password }`. Risposta: access token + utente; imposta cookie HttpOnly `rt` per refresh.

**POST /auth/refresh** Pubblica. Usa il cookie HttpOnly `rt`. Scopo: rilascio nuovo access token.

**POST /auth/logout** Protetta (`requireAuth`). Scopo: invalida la sessione corrente e cancella il cookie refresh.

**POST /auth/logout-all** Protetta (`requireAuth`). Scopo: revoca tutte le sessioni dell'utente.

**GET /auth/me** Protetta (`requireAuth`). Scopo: dati utente corrente.

Esempio richiesta/risposta login:

```
1 POST api/auth/login
2 // richiesta
3 {
4     "email": "sabs@esempio.it",
5     "password": "hellokitty"
6 }
7 // risposta
8 {
9     "accessToken": "<jwt>",
10    "user":
11    {
12        "id": 100, "email": "sabs@esempio.it", "role": "PASSEGGERO",
13        "foto": "sabs.jpg"
14    }
15 }
```

Esempio richiesta/risposta register:

```
1 POST /api/auth/register
2 // richiesta
3 {
```

```

4   "email": "sabs@esempio.it",
5   "password": "hellokitty",
6   "dati": {
7     "nome": "Matilde",
8     "cognome": "Esposito",
9     "codiceFiscale": "SPSMLD00A41L407X",
10    "dataNascita": "2000-01-01",
11    "sesto": "F"
12  }
13}
14// risposta
15{
16  "accessToken": "<jwt>", "user": { "id": 100,
17  "email": "sabs@esempio.it", "role": "PASSEGGERO", "foto":
→ "sabs.jpg"}
18}

```

### 3.2 Aeroporti (/aeroporti)

GET /aeroporti/list Pubblica. Scopo: elenco aeroporti raggruppati per nazione.

### 3.3 Ricerca soluzioni (/soluzioni)

GET /soluzioni/ricerca Pubblica. query: partenza, arrivo, data\_andata, data Ritorno (opz). Scopo: calcolare itinerari andata/ritorno.

Esempi risposte:

```

1 // risposta
2 // solo andata
3 [
4   {
5     "durata_totale": "2h 15m",
6     "voli": [
7       {
8         "numero": "AZ-123456",
9         "compagnia": "ITA Airways",
10        "partenza": "FCO",
11        "arrivo": "AMS",
12        "citta_partenza": "Roma",
13        "citta_arrivo": "Amsterdam",
14        "ora_partenza": "2025-10-02T08:15:00",
15        "ora_arrivo": "2025-10-02T10:30:00",
16        "modello": "A320",
17        "distanza": 1295
18      }
19    ]
20  }
21]

```

```

22
23 // risposta
24 // andata e ritorno
25 {
26   "andata": [ { "durata_totale": "2h 15m", "voli": [ /* ... */ ] } ],
27   "ritorno": [ { "durata_totale": "2h 05m", "voli": [ /* ... */ ] } ]
28 }

```

### 3.4 Prenotazione posti (/booking)

Tutte protette (`requireAuth`) e riservate ai ruoli PASSEGGERO (`requireRole`).

**GET /booking/configuration** query: `nome` (modello). Scopo: configurazione posti per modello.

**GET /booking/seats** query: `volo`. Scopo: posti già occupati/riservati per un volo.

**POST /booking/seats/reserve** body: array di biglietti da trattenere 15 min. Scopo: trattenuta temporanea dei posti.

Esempi:

```

1 GET /api/booking/configuration?nome=Boeing 737-800
2 // risposta
3 {
4   "nome": "Boeing 737-800",
5   "totale_posti": 186,
6   "posti_economy": 162,
7   "posti_business": 18,
8   "posti_first": 6,
9   "layout": "3-3"
10 }
11
12 GET /api/booking/seats?volo=ROUTE-000001
13 // risposta
14 { "occupied": ["12A", "12B", "14C"] }
15
16 POST /api/booking/seats/reserve
17 // richiesta
18 [
19   { "volo": "ROUTE-000001", "posto": "12A", "classe": "economy",
20     "prezzo": 129.99, "nome": "Matilde", "cognome": "Esposito",
21     "bagagli": 1 },
22   { "volo": "ROUTE-000001", "posto": "12B", "classe": "economy",
23     "prezzo": 129.99, "nome": "Chiara", "cognome": "Taranto",
24     "bagagli": 0 }
25 ]
26 // risposta
27 { "success": true }

```

### 3.5 Checkout e pagamenti (`/checkout`)

Tutte protette (`requireAuth`) e riservate a PASSEGGERO.

**POST /checkout/insert-tickets** body: lista biglietti confermati. Scopo: conferma definitiva (rimuove scadenza).

**POST /checkout/create-payment-intent** body: importo/valuta/tipi. Scopo: PaymentIntent Stripe per pagamento.

**GET /checkout/payment-intent/:pi\_id** params: `pi_id`. Scopo: stato PaymentIntent.

**POST /checkout/stripe-webhook** Pubblica. body: raw (firma Stripe verificata). Scopo: ricezione eventi Stripe.

Esempi:

```
1 POST /api/checkout/insert-tickets
2 // richiesta
3 [
4     { "flightNumber": "ROUTE-000001", "seatNumber": "12A", "seatClass":
5         "economy", "seatPrice": 129.99, "firstName": "Matilde",
6         "lastName": "Esposito", "extraBags": 1 },
7     { "flightNumber": "ROUTE-000001", "seatNumber": "12B", "seatClass":
8         "economy", "seatPrice": 129.99, "firstName": "Chiara",
9         "lastName": "Taranto", "extraBags": 0 }
10 ]
11 // risposta
12 { "message": "Checkout avvenuto con successo" }

13 POST /api/checkout/create-payment-intent
14 // richiesta
15 { "amount": 25999, "currency": "eur", "orderId": "ORD-123",
16     "customerEmail": "sabs@esempio.it" }
17 // risposta
18 { "clientSecret": "pi_..._secret_...", "paymentIntentId": "pi_..." }

19 GET /api/checkout/payment-intent/pi_abc123
20 // risposta
21 { "status": "succeeded" }
```

### 3.6 Area passeggero (`/passeggero`)

Tutte protette (`requireAuth`) e ruolo PASSEGGERO.

**GET /passeggero/profile** Scopo: profilo utente.

**POST /passeggero/update/foto** body (multipart): campo `file` (max 5MB, solo immagini). Scopo: aggiorna foto profilo.

**GET /passeggero/reservations** Scopo: elenco prenotazioni utente.

**GET /passeggero/statistics** Scopo: statistiche viaggio.

**PUT /passeggero/aggiorna-email** body: nuova email. Scopo: aggiornare email.

**PUT /passeggero/aggiorna-password** body: vecchia+nuova password. Scopo: aggiornare password.

**POST /passeggero/stripe/setup-intent** Scopo: crea SetupIntent.

**GET /passeggero/stripe/payment-methods** Scopo: lista metodi di pagamento.

**DELETE /passeggero/stripe/payment-methods/:pmId** params: pmId. Scopo: rimuove metodo.

Esempi:

```
1 GET /api/passeggero/profile
2 // risposta
3 { "id": 120, "email": "sabs@esempio.it", "nome": "Matilde", "cognome":
4   → "Esposito", "codice_fiscale": "SPSMLD00A41L407X", "data_nascita":
5   → "2000-01-01", "sesso": "F", "foto": "sabs.jpg" }
6
7 GET /api/passeggero/reservations
8 // risposta
9 [
10   {
11     "firstName": "Matilde", "lastName": "Esposito", "flightNumber":
12       → "ROUTE-000001", "from": "FCO", "to": "AMS", "cityFrom": "Roma",
13       → "cityTo": "Amsterdam", "departureDate": "2025-10-02",
14       → "departureTime": "08:15", "seatNumber": "12A", "seatClass":
15       → "economy", "seatPrice": 129.99, "extraBags": 1
16   }
17 ]
18
19 GET /api/passeggero/statistics
20 // risposta
21 { "totalFlights": 8, "visitedCountries": 5, "kilometersFlown": 12450,
22   → "flightsThisYear": 3 }
23
24 PUT /api/passeggero/aggiorna-email
25 // richiesta
26 { "email": "sabri@esempio.it" }
27 // risposta
28 { "message": "Email aggiornata con successo" }
29
30 PUT /api/passeggero/aggiorna-password
31 // richiesta
32 { "passwordAttuale": "hellokitty", "nuovaPassword": "mymelody" }
33 // risposta
34 { "message": "Password aggiornata con successo" }
35
36 POST /api/passeggero/stripe/setup-intent
37 // risposta
```

```

29 { "clientSecret": "seti_..._secret_..." }
30
31 GET /api/passeggero/stripe/payment-methods
32 // risposta
33 [ { "id": "pm_123", "brand": "visa", "last4": "4242", "exp_month": 12,
34   → "exp_year": 2030 } ]
35
36 DELETE /api/passeggero/stripe/payment-methods/pm_123
37 // risposta
{ "message": "Metodo rimosso", "id": "pm_123" }

```

### 3.7 Area compagnia (/compagnia)

Protette (`requireAuth`) e ruolo COMPAGNIA, salvo note.

**GET /compagnia/profile** Scopo: profilo compagnia.

**POST /compagnia/setup** body: dati azienda e cambio password. Scopo: setup iniziale.

**GET /compagnia/uploads/compagnie/:filename** Pubblica. Scopo: servire immagini logo.

**GET /compagnia/statistics** Scopo: statistiche lato compagnia.

**GET /compagnia/aircrafts** Scopo: elenco aerei della compagnia.

**POST /compagnia/aircrafts** body: nuovo aereo. Scopo: aggiungere aereo.

**DELETE /compagnia/aircrafts/:numero** params: `numero`. Scopo: rimuovere aereo.

**GET /compagnia/models** Scopo: elenco modelli disponibili.

**GET /compagnia/routes** Scopo: elenco tratte.

**GET /compagnia/routes/best** Scopo: migliori tratte per resa.

**POST /compagnia/routes** body: nuova tratta. Scopo: aggiungere tratta.

**DELETE /compagnia/routes/:numero** params: `numero`. Scopo: rimuovere tratta.

**GET /compagnia/flights** Scopo: elenco voli programmati.

**POST /compagnia/flights** body: nuovi voli. Scopo: aggiungere voli.

Esempio creazione aereo:

```

1 POST /api/compagnia/aircrafts
2 // richiesta
3 {
4     "modello": "Boeing 737-800"
5 }
6 // risposta
7 { "numero": "AZ-B738-001", "modello": "Boeing 737-800" }

```

Altri esempi Compagnia:

```
1 GET /api/compagnia/profile
2 // risposta
3 { "nome": "ITA Airways", "codice_iata": "AZ", "contatto": "+39 06 ...",
4   "nazione": "Italia", "foto": "az.png" }
5
6 GET /api/compagnia/routes
7 // risposta
8 [ { "numero": "RT-AZ-001", "partenza": "FCO", "arrivo": "AMS",
9   "durata_min": 135, "lunghezza_km": 1295, "partenza_nome": "Roma
10    Fiumicino", "arrivo_nome": "Schiphol" } ]
11
12 GET /api/compagnia/flights
13 // risposta
14 [ { "numero": "RT-AZ-001-000001", "partenza": "2025-10-02 08:15",
15   "arrivo": "2025-10-02 10:30", "tratta_id": "RT-AZ-001", "aereo_nome":
16   "A320", "posti_disponibili": 42 } ]
17
18 POST /api/compagnia/routes
19 // richiesta
20 { "numero": "RT-AZ-002", "partenza": "FCO", "arrivo": "CDG",
21   "durata_min": 120, "lunghezza_km": 1105 }
22 // risposta
23 { "message": "Tratta aggiunta con successo" }
24
25 POST /api/compagnia/flights
26 // richiesta
27 { "routeNumber": "RT-AZ-001", "aircraftNumber": "AZ-A320-001",
28   "frequency": "giornaliero", "departureTime": "08:15", "startDate":
29   "2025-10-01", "weeksCount": 2 }
30 // risposta
31 { "message": "Voli aggiunti con successo", "created":
32   ["RT-AZ-001-000123", "RT-AZ-001-000124", "..."] }
```

### 3.8 Area amministratore (/admin)

Protette (requireAuth) e ruolo ADMIN.

**GET /admin/compagnie** Scopo: elenco compagnie.

**GET /admin/passeggeri** Scopo: elenco passeggeri.

**DELETE /admin/compagnie/:id** params: id. Scopo: rimuovi compagnia.

**DELETE /admin/passeggeri/:id** params: id. Scopo: rimuovi passeggero.

**POST /admin/aggiungi** body (multipart/form-data): campi **email** (string), **password** (string), **file** (logo). Scopo: crea account compagnia con foto. Risposta: 204/void (nessun body).

**GET /admin/compagnie/attesa** Scopo: elenco richieste in attesa.

**DELETE /admin/compagnie/attesa/:id** params: id. Scopo: rimuovi richiesta.

```
1 GET /api/admin/compagnie
2 // risposta
3 [ { "utente": 12, "nome": "ITA Airways", "email": "info@itaairways.com",
4   → "nazione": "Italia" } ]
5
6 GET /api/admin/passeggeri
7 // risposta
8 [ { "utente": 100, "email": "sabs@esempio.it", "nome": "Matilde",
9   → "cognome": "Esposito", "foto": "sabs" } ]
10
11 POST /api/admin/aggiungi
12 // richiesta
13 email=info@itaairways.com, password=S3greta!, file=<logo.png>
14 // risposta
15 { "message": "Compagnia creata con successo!", "id": 10 }
16
17 GET /api/admin/compagnie/attesa
18 // risposta
19 [ { "utente": 10, "email": "info@compagnia.com" } ]
20
21 DELETE /api/admin/compagnie/attesa/10
22 // risposta
23 204 No Content
```

## 4 Autenticazione e gestione sessioni

Il sistema utilizza JWT con due token:

- **Access token** firmato con segreto ACCESS\_SECRET, breve scadenza (default 5 min), inviato come Bearer nelle richieste; validato da `requireAuth`.
- **Refresh token** firmato con REFRESH\_SECRET, lunga scadenza (default 7 giorni), salvato in cookie HttpOnly (path /api/auth). Il suo “`jti`” è registrato in tabella `sessioni` per consentire revoca e controllo scadenza.

Il backend offre anche la rotazione opzionale dei segreti all'avvio (env ROTATE\_SECRETS\_ON\_START=1).

### 4.1 Flusso client

1. Login/Register → ricezione dell'access token (in `localStorage`) e del refresh token (in cookie `HttpOnly`).
2. L'`HttpInterceptor` allega automaticamente l'access token e, in caso di errore 401 (scadenza o invalidità), invoca `/auth/refresh` una sola volta, aggiorna l'access token e ritenta la richiesta originale.
3. Logout → rimozione dell'access token, cancellazione del cookie ed eliminazione della sessione nel DB tramite `jti`; `logout-all` → rimozione dell'access token, cancellazione del cookie ed eliminazione di tutte le sessioni nel DB per l'utente interessato.

### 4.2 Guardie di routing (Angular)

`extbfauth-role.guard.ts` protegge le rotte in base al ruolo richiesto (`data.role`). Se c'è un utente già in memoria, verifica il ruolo; se esiste solo l'access token, chiama `AuthService.me$()` per ottenere i dati utente e quindi valida il ruolo. In caso negativo, effettua redirect alla home. Questo garantisce che le pagine come `/passeggero`, `/aerolinea` e `/admin` siano raggiungibili solo da utenti con ruolo coerente.

### 4.3 HttpInterceptor: Authorization e refresh

L'interceptor (snippet ridotto di seguito) si occupa di:

- Allegare a ogni richiesta API protetta l'header:

```
Authorization: Bearer <access-token>
```

letto dal `localStorage`.

- Intercettare risposte 401/ACCESS\_TOKEN\_EXPIRED da `requireAuth` e avviare un flusso di `refresh` centralizzato (debounce/singola esecuzione concorrente) verso `/api/auth/refresh` con `withCredentials: true` (il cookie `HttpOnly rt` è inviato automaticamente dal browser).
- Aggiornare l'access token su successo e ritentare trasparentemente la richiesta originale.

L'access token ha durata breve per limitare l'impatto in caso di esfiltrazione; il refresh token non è accessibile a JS (HttpOnly) e non viene inviato a tutte le rotte, ma **solo** a quelle sotto `/api/auth` grazie all'attributo `path=/api/auth` del cookie.

**Scoping del cookie e rotte escluse** Il cookie HttpOnly del refresh è **scambiato solo** con rotte di autenticazione grazie a `path=/api/auth`. Nel codice dell'interceptor è presente una lista di rotte escluse:

```
const EXCLUDED = [
  '/api/auth/register',
  '/api/auth/login',
  '/api/auth/refresh',
  '/api/auth/logout',
];
```

Per tali chiamate non si allega l'header `Authorization` e non si attiva il flusso di refresh: sono endpoint di bootstrap dell'autenticazione che usano `withCredentials: true` per *ricevere/mandare* il refresh token via cookie, senza richiedere un access token già valido. In questo modo l'utente non percepisce differenze: il rinnovo avviene in background e le richieste applicative vengono ritentate automaticamente.

### Snippet (TypeScript, semplificato)

```
1 import { inject } from '@angular/core';
2 import { HttpInterceptorFn, HttpErrorResponse } from
3   '@angular/common/http';
4 import { AuthService } from '../services/auth';
5 import { Subject, throwError } from 'rxjs';
6 import { catchError, switchMap, take } from 'rxjs/operators';
7
8 let isRefreshing = false;
9 const refreshDone$ = new Subject<boolean>();
10
11 export const authInterceptor: HttpInterceptorFn = (req, next) => {
12   const auth = inject(AuthService);
13   const attach = (token: string | null) =>
14     token ? req.clone({ setHeaders: { Authorization: `Bearer ${token}` } })
15       : req;
16
17   return next(attach(auth.token)).pipe(
18     catchError((err: HttpErrorResponse) => {
19       if (err.status !== 401) return throwError(() => err);
20
21       if (!isRefreshing) {
22         isRefreshing = true;
23         return auth.refresh().pipe(
24           switchMap(() => {
25             isRefreshing = false; refreshDone$.next(true);
26             return next(attach(auth.token));
27           }),
28           catchError(e => {
29             isRefreshing = false; refreshDone$.next(false);
30             auth.logout().subscribe();
31             return throwError(() => e);
32           })
33         );
34       }
35     })
36   );
37 }
```

```

30         })
31     );
32   }
33
34   return refreshDone$.pipe(
35     take(1),
36     switchMap(ok => ok ? next(attach(auth.token))
37               : throwError(() => err))
38   );
39 }
40 );
41 };

```

## 4.4 Middleware server: requireAuth e requireRole

`extbfrequireAuth` legge l'header Bearer, verifica la firma con `verifyAccessToken` e, se valido, popola `req.user` con il payload JWT (`sub=id, role`). In caso di scadenza genera 401 con codice `ACCESS_TOKEN_EXPIRED`; in caso di token non valido, 401 generico.  
`extbfrequireRole(...roles)` verifica che `req.user.role` sia tra quelli ammessi, altrimenti risponde 403. In questo modo il controller riceve già `req.user.sub` e `req.user.role` in modo affidabile, senza dover rileggere parametri lato client.

## 4.5 Tabella sessioni come whitelist dei refresh

Alla creazione di un refresh token si memorizzano `jti` (JWT ID), `utente`, scadenza nella tabella `sessioni`. Il flusso `/auth/refresh`:

1. Verifica la firma del refresh token (`verifyRefreshToken`).
2. Estraie il `jti` e controlla in `sessioni` che la sessione non sia revocata (`revocato=false`) e non sia scaduta (`scadenza > now()`).
3. Emette un nuovo access token e ritorna i dati utente.

**Revoca:**

- `/auth/logout`: invalida solo la sessione corrente (revoca per `jti`).
- `/auth/logout-all`: revoca tutte le sessioni dell'utente (`UPDATE sessioni SET revocato=TRUE WHERE utente=...`).

## 4.6 Rotazione segreti (ROTATE\_SECRETS\_ON\_START)

- **Sviluppo:** tipicamente `ROTATE_SECRETS_ON_START=0` per evitare di invalidare continuamente le sessioni e velocizzare il testing.
- **Produzione:** con `ROTATE_SECRETS_ON_START=1` il server genera chiavi casuali ad ogni riavvio (`generateRandomSecret()`) per **entrambe** le firme (access e refresh) e forza la revoca di tutte le sessioni (`UPDATE sessioni SET revocato=TRUE`). Questo invalida i token sia per *cambio firma* sia per *revoca server-side*.

## 4.7 Sicurezza: scelte e mitigazioni

- **Separazione token:** access token in `localStorage` (breve TTL) e refresh token in cookie `HttpOnly` con `sameSite=lax` e `path=/api/auth`. Così si mitiga **XSS** (il refresh non è leggibile da JS) e si riduce la superficie **CSRF** (il cookie è inviato solo verso rotte di refresh).
- **CORS** con origine configurabile (`CORS_ORIGIN`) e `credentials=true` per controllare chi può invocare le API con cookie.
- **Input validation e SQL injection:** tutte le query usano `pool.query(..., [parametri])` con binding parametrico; ci sono validazioni esplicite (es. `setupCompany`) e controlli semantici nei controller.
- **Password hashing:** `bcrypt` con cost 12 per memorizzazione sicura delle credenziali.
- **Upload sicuri:** `multer` limita tipo MIME a immagini e dimensione max (5 MB), nomi randomizzati e directory segregate; i file sono serviti da percorsi dedicati.
- **Principio del minimo privilegio:** middleware `requireRole` restringe l'accesso alle sole funzioni pertinenti (es. rotte compagnia/admin).
- **Nessun token in URL:** i token non compaiono in query string o percorsi, evitando leakage in log e referrer.
- **Cookie scoping:** `path=/api/auth` impedisce l'invio del refresh verso altre rotte, evitando *over-sharing* del token.
- **Short-lived access token:** riduce impatto di furto del Bearer; l'interceptor gestisce rinnovo trasparente.

## File e risorse coinvolte nell'autenticazione

L'intero meccanismo di autenticazione/autorizzazione è realizzato tramite le tabelle **utenti** e **sессии** del DB e i seguenti file lato client e server:

- **Server**
  - `server/src/utils/jwt.ts`
  - `server/src/middleware/auth.ts`
  - `server/src/middleware/role.ts`
  - `server/src/controllers/authController.ts`
  - `server/src/routes/authRoutes.ts`
  - `server/src/types/express.d.ts`
  - `server/src/server.ts`
- **Client**
  - `client/src/app/services/auth.ts`
  - `client/src/app/guard/auth.interceptor.ts`
  - `client/src/app/guard/auth-role.guard.ts`
  - `client/src/app/app.routes.ts`
  - `client/src/app/app.config.ts`

## 5 Front end Angular

### 5.1 Struttura

L'interfaccia è una SPA basata su componenti. Il Router decide quale *pagina* mostrare in base al percorso, iniettando la vista dentro `<router-outlet>`. Parti riutilizzabili (Navbar, Footer, ...) sono componenti condivisi inclusi nei template delle pagine.

#### Routes (`app.routes.ts`)

`/home` Pagina principale (pubblica)

`/voli` Ricerca/Lista voli (pubblica)

`/dettagli` Sezione per l'inserimento dei dati dei passeggeri coinvolti nell'acquisto (pubblica).

`/posti` Selezione posti per il volo (protetta, ruolo PASSEGGERO)

`/checkout` Checkout e pagamento (protetta, ruolo PASSEGGERO)

`/passeggero` Area riservata passeggero (protetta, ruolo PASSEGGERO)

`/aerolinea` Area riservata compagnia aerea (protetta, ruolo COMPAGNIA)

`/admin` Pannello amministratore (protetta, ruolo ADMIN)

#### Guardie e Interceptor

- `auth-role.guard.ts`: consente l'accesso alle rotte riservate solo se l'utente ha il ruolo richiesto (dato da rotta `data.role`). In caso contrario effettua un redirect sicuro.
- `auth.interceptor.ts`: allega l'header `Authorization: Bearer <token>` alle chiamate API; in caso di 401 per scadenza o invalidità gestisce un refresh centralizzato dell'access token e ritenta la richiesta originale.

#### Servizi (Services)

- `auth`: login, registrazione, me, refresh, logout e logout all.
- `aeroporti`: elenco aeroporti per nazione.
- `soluzioni`: ricerca itinerari (andata/ritorno).
- `booking`: configurazione posti per modello, posti occupati per volo, trattenuta temporanea.
- `checkout`: inserimento biglietti confermati, creazione/stato PaymentIntent Stripe.
- `passeggero`: profilo, foto profilo, prenotazioni, statistiche, gestione email/password, metodi di pagamento.
- `aerolinea`: profilo compagnia, setup iniziale, aerei, modelli, tratte, voli, statistiche.
- `admin`: gestione compagnie/passeggeri, inviti/aggiunta compagnie, richieste in attesa.

#### Componenti implementate

- **Pagine**: Home, Voli, Dettagli, Posti, Checkout, Passeggero, Aerolinea, Admin.
- **Condivisi**: Navbar, Footer, Login/Registrazione, Ticket/Biglietto, componenti UI riutilizzabili (card, form, dialog) stilizzati con Tailwind CSS e daisyUI.

## 6 Screenshot e flussi per ruolo

### 6.1 Passeggero

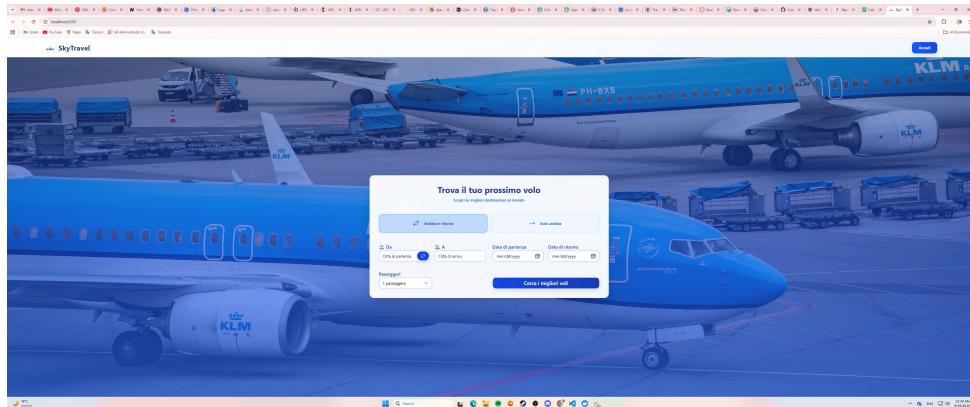


Figura 1: Pagina principale

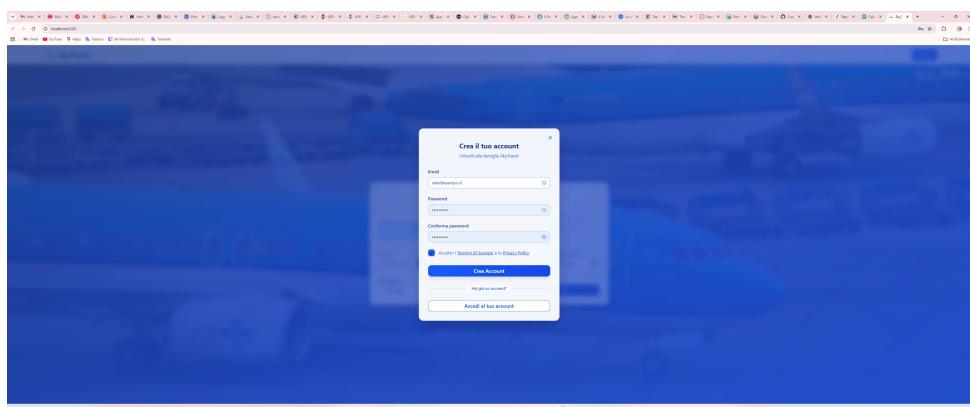


Figura 2: Registrazione dell'account (email e password)

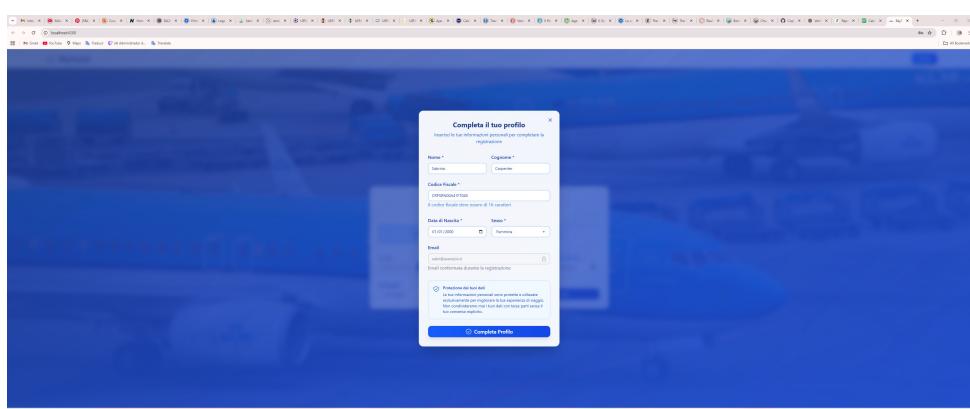


Figura 3: Inserimento dei dati del passeggero

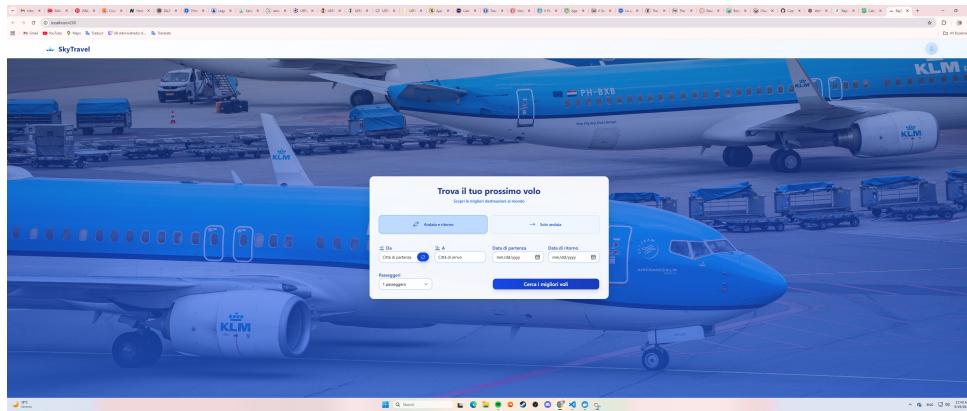


Figura 4: Account autenticato immediatamente dopo la registrazione

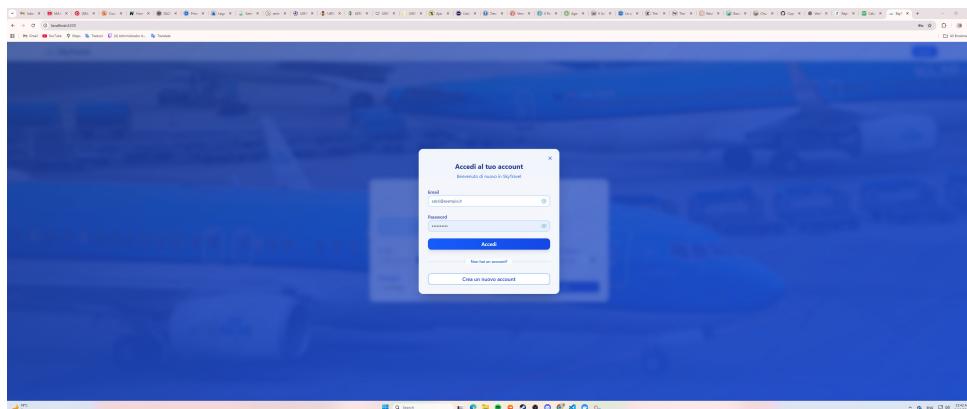


Figura 5: Pagina di accesso del passeggero

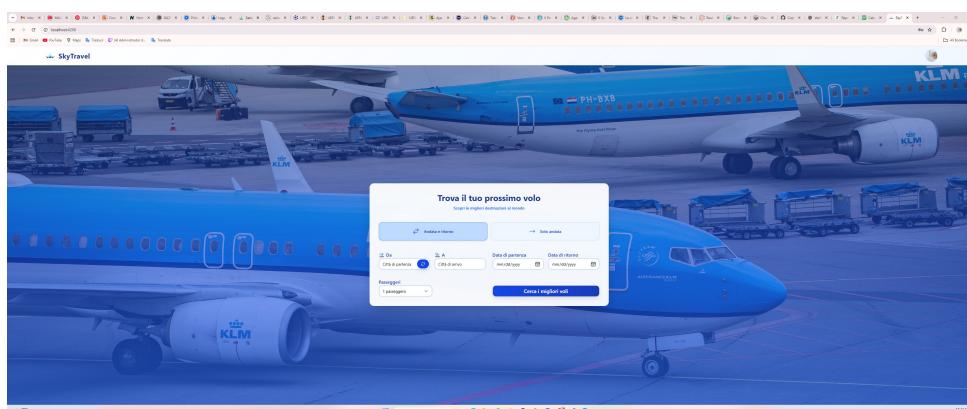


Figura 6: Account autenticato immediatamente dopo l'accesso

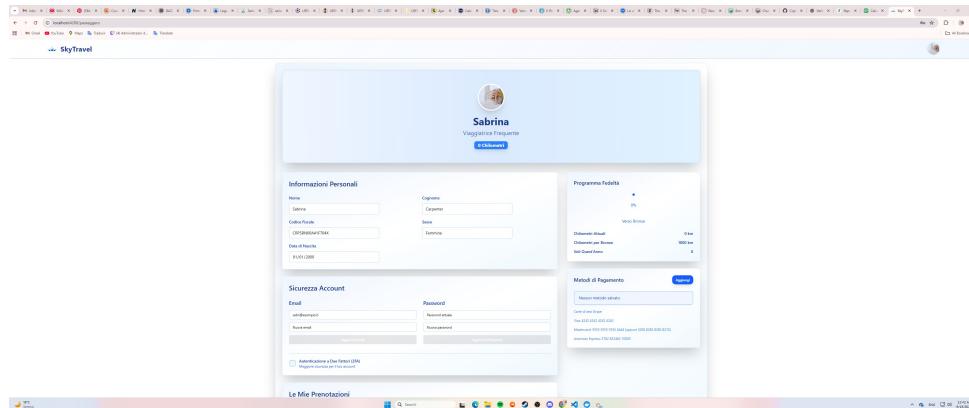


Figura 7: Schermata del profilo del passeggero

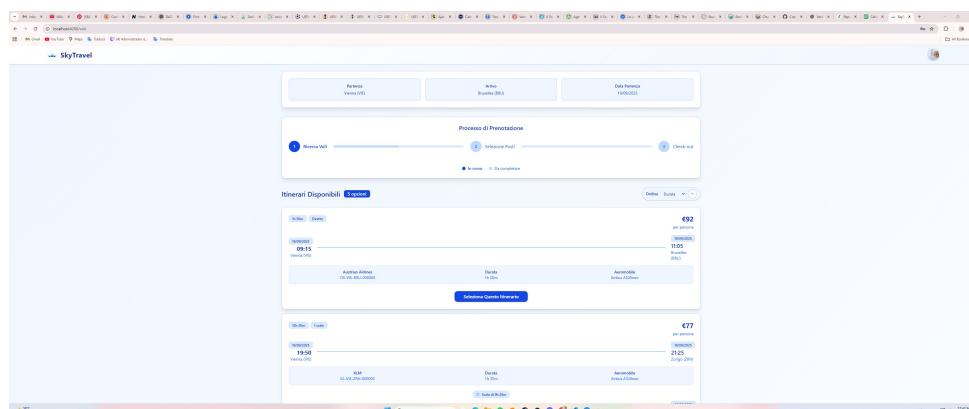


Figura 8: Schermata per la visualizzazione dei voli disponibili per la tratta scelta

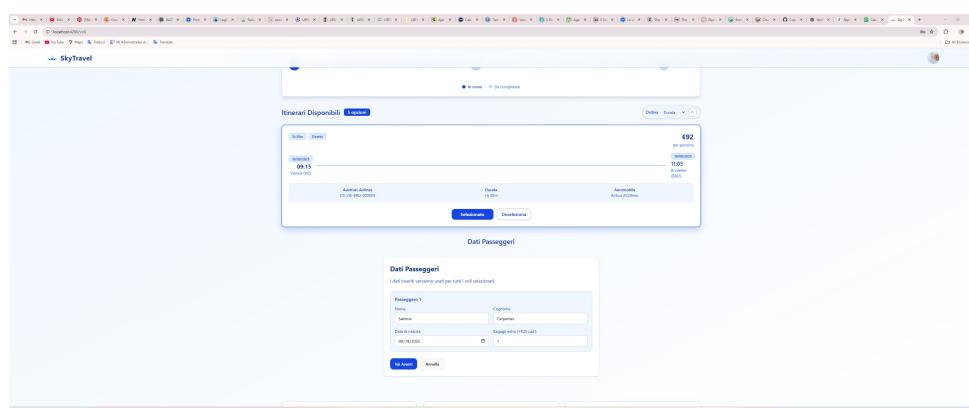


Figura 9: Inserimento dei dati dei passeggeri interessati

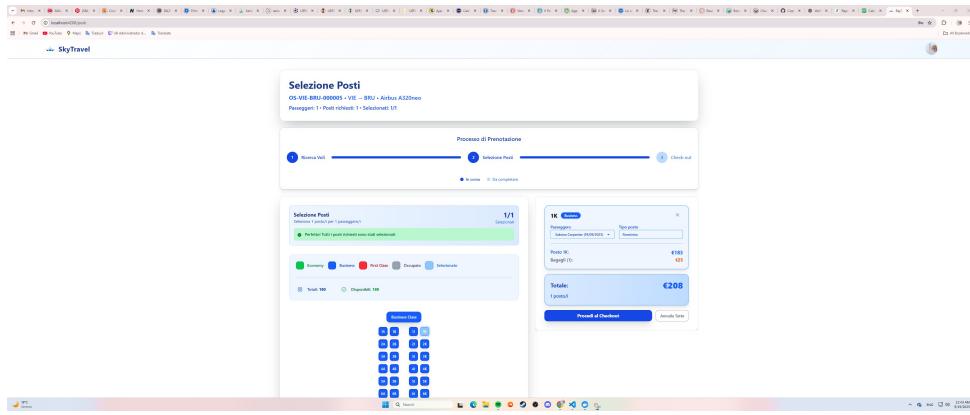


Figura 10: Selezione dei posti per il volo

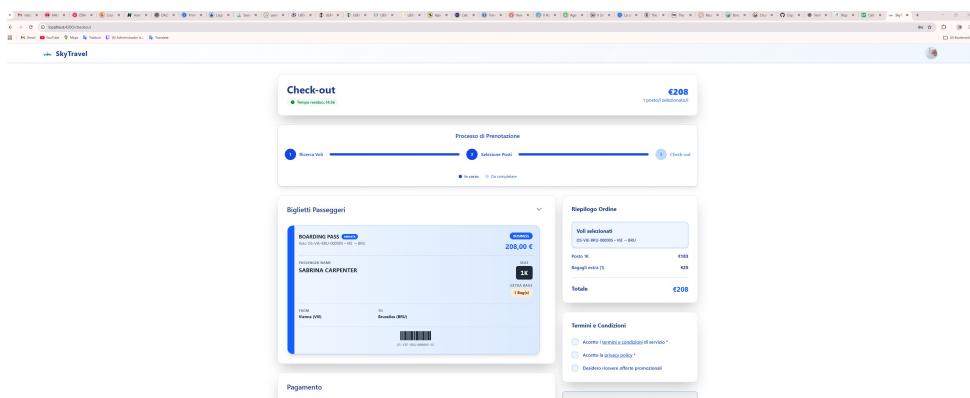


Figura 11: Schermata di checkout per procedere al pagamento

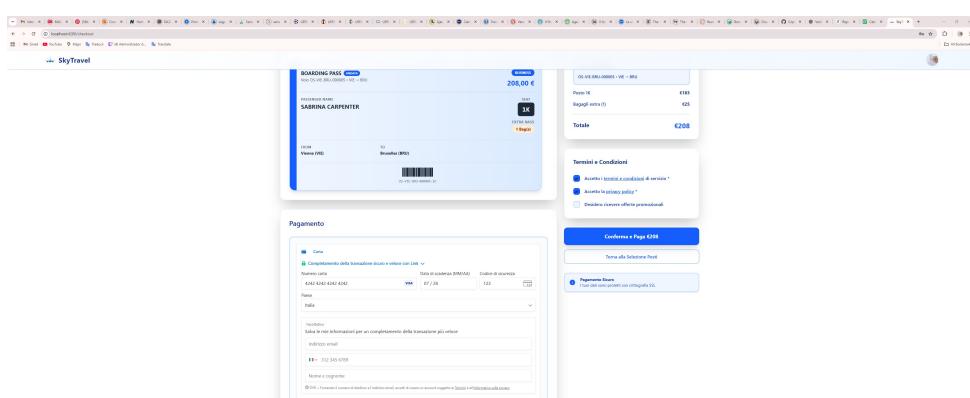


Figura 12: Inserimento dei dati della carta

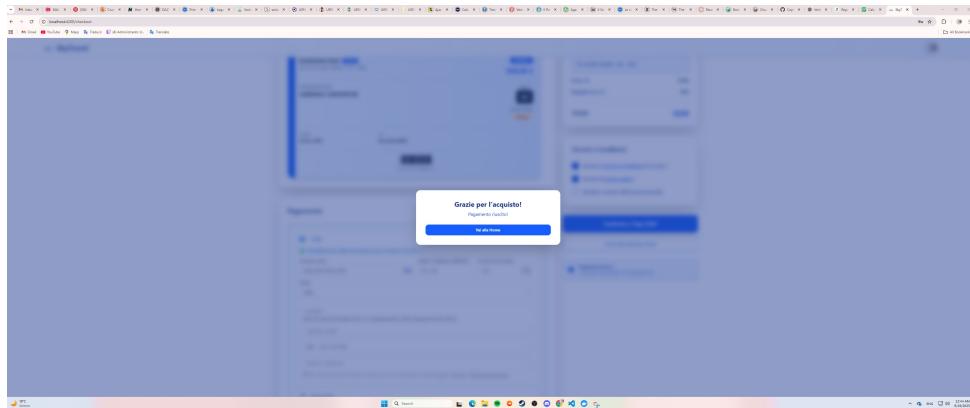


Figura 13: Conferma di prenotazione avvenuta con successo

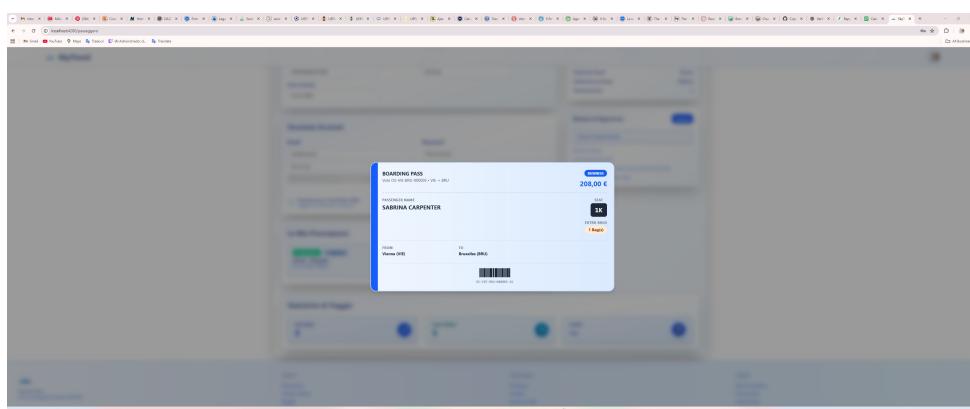


Figura 14: Visualizzazione dei biglietti acquistati

## 6.2 Admin e compagnia aerea

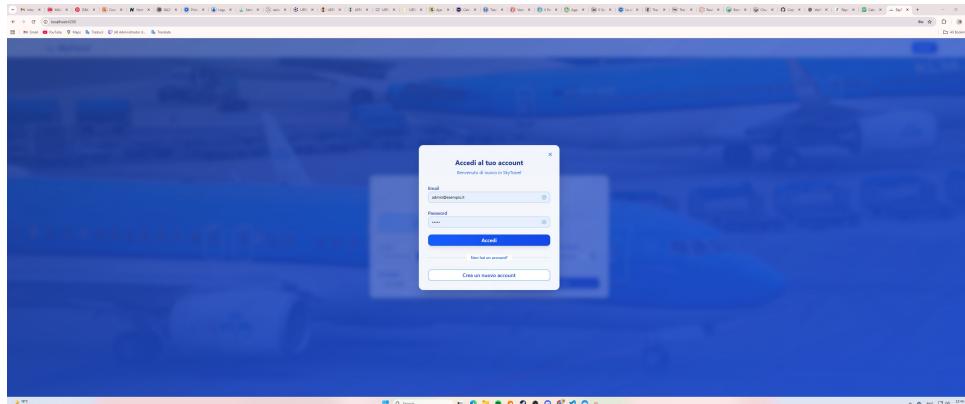


Figura 15: Pagina di accesso del admin

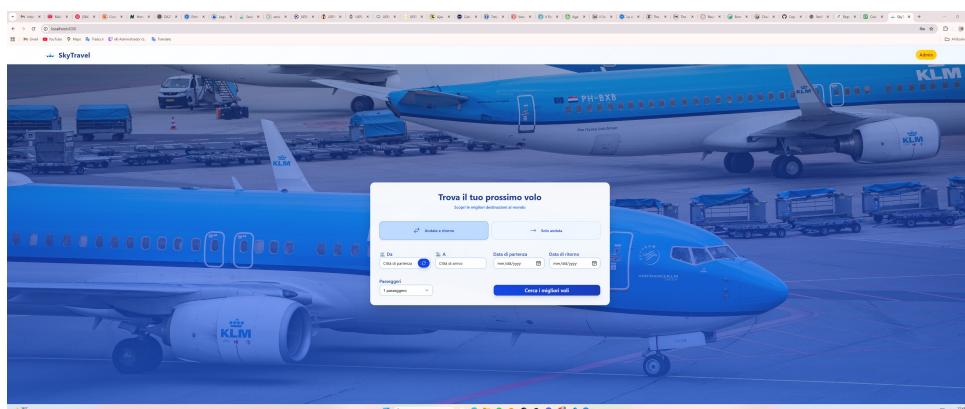


Figura 16: Account autenticato immediatamente dopo l'accesso

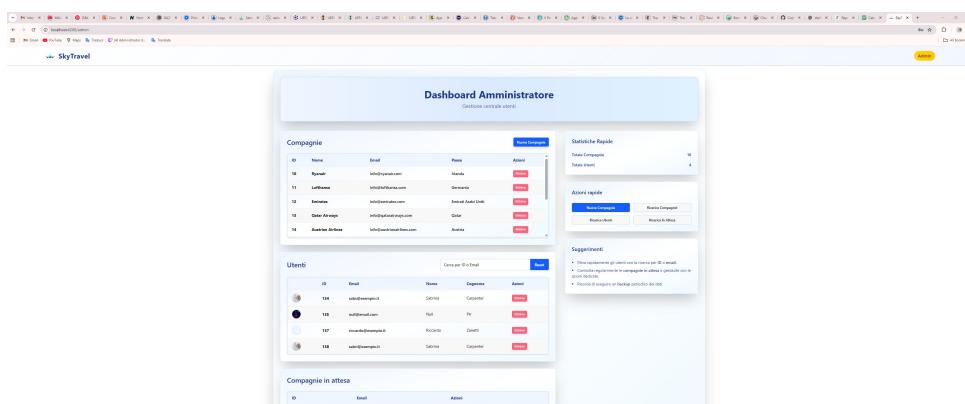


Figura 17: Schermata della dashboard dell'admin

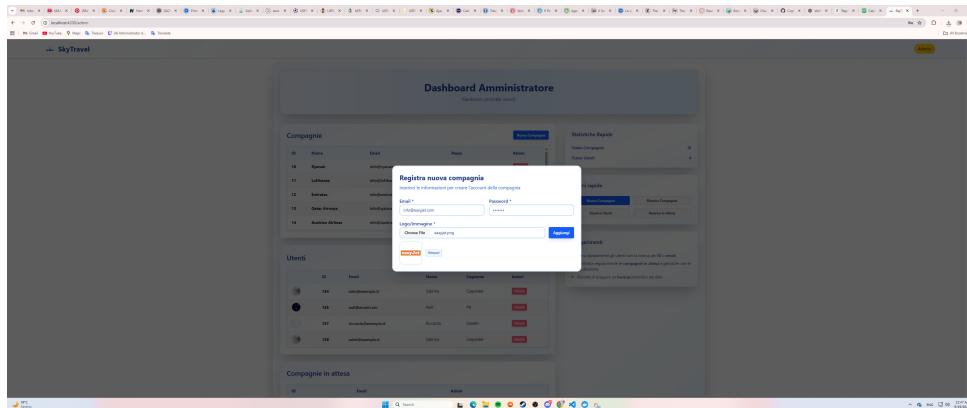


Figura 18: Schermata di aggiunta di una compagnia aerea

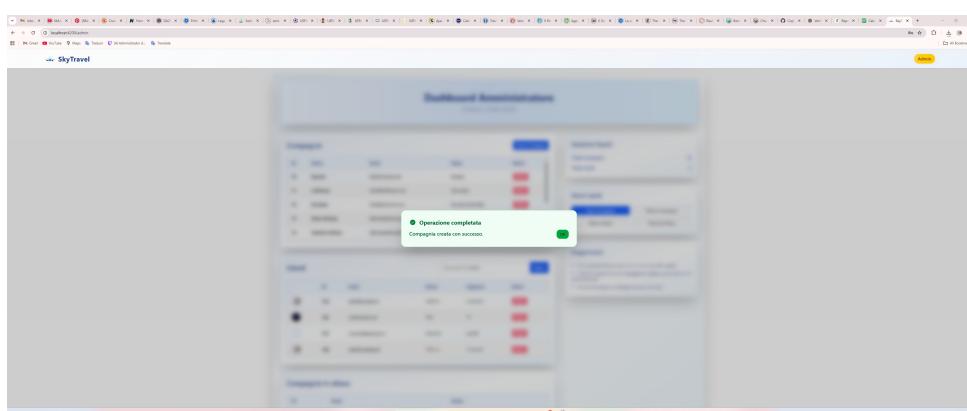


Figura 19: Popup di conferma dell'inserimento

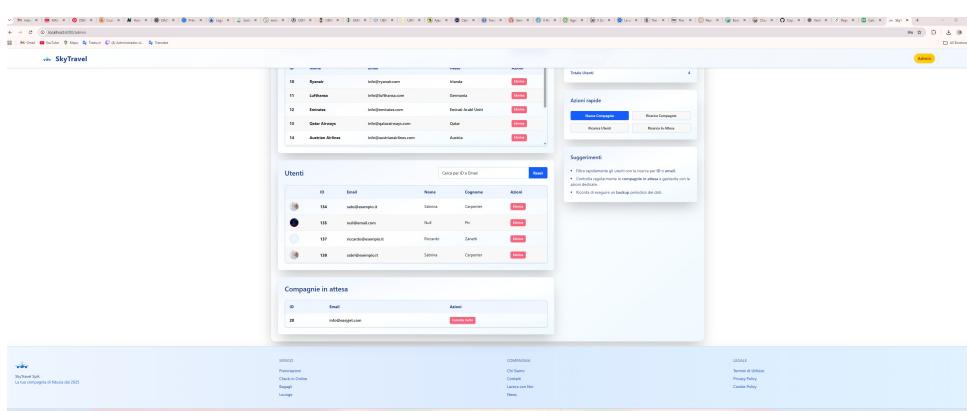


Figura 20: Visualizzazione delle compagnie aeree con invito in attesa

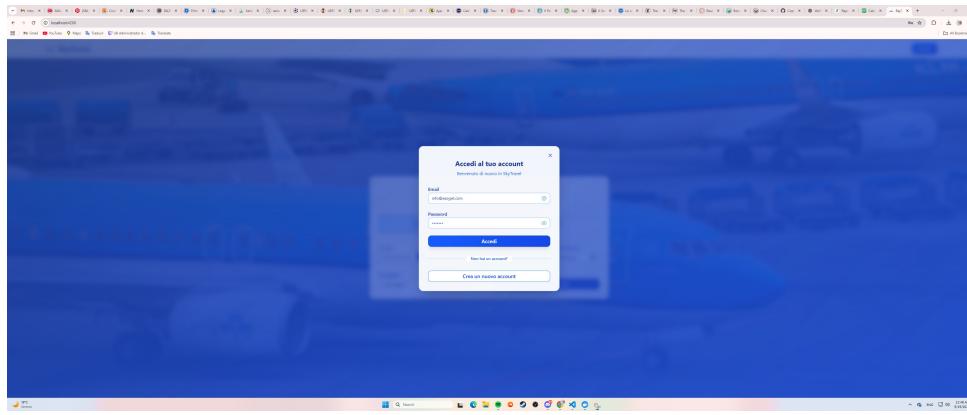


Figura 21: Pagina di accesso della compagnie aerea

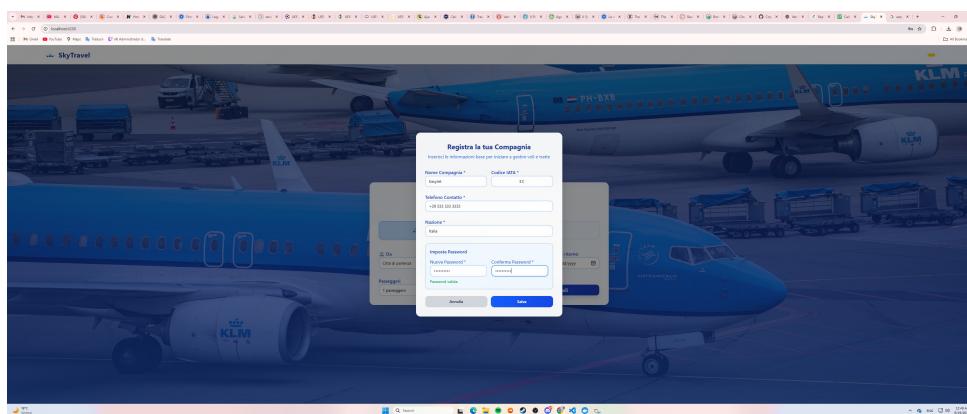


Figura 22: Schermata per l'inserimento dei dati e il cambio password dell'aerolinea

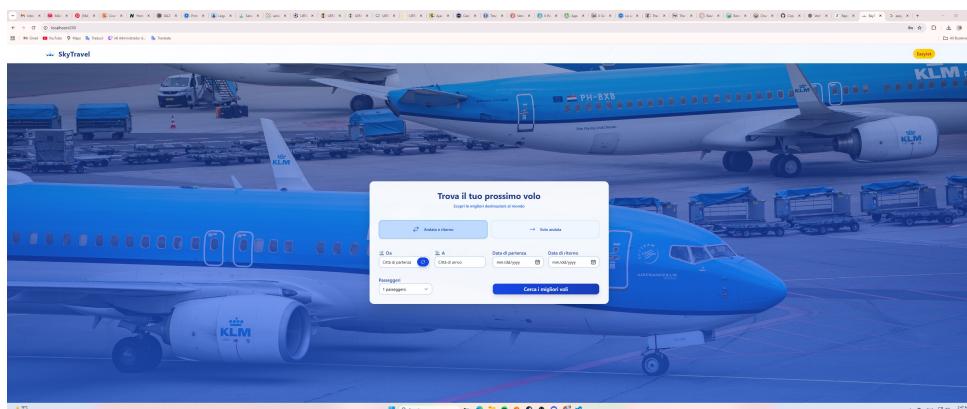


Figura 23: Account autenticato immediatamente dopo l'accesso

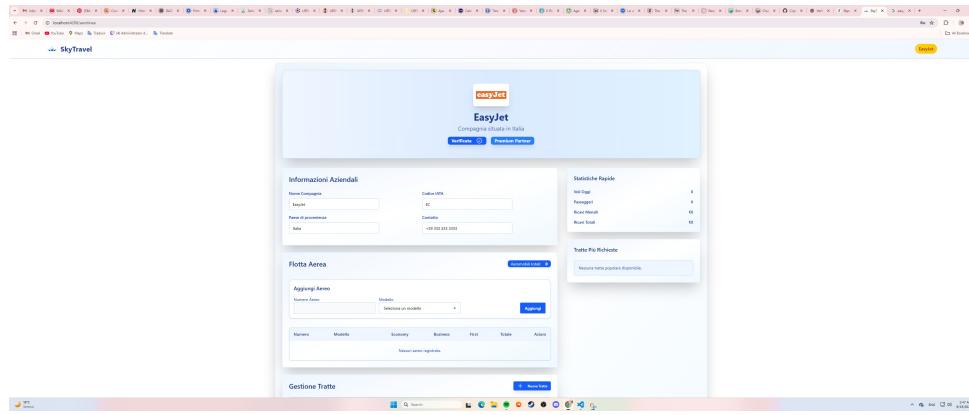


Figura 24: Schermata del profilo dell'aerolinea

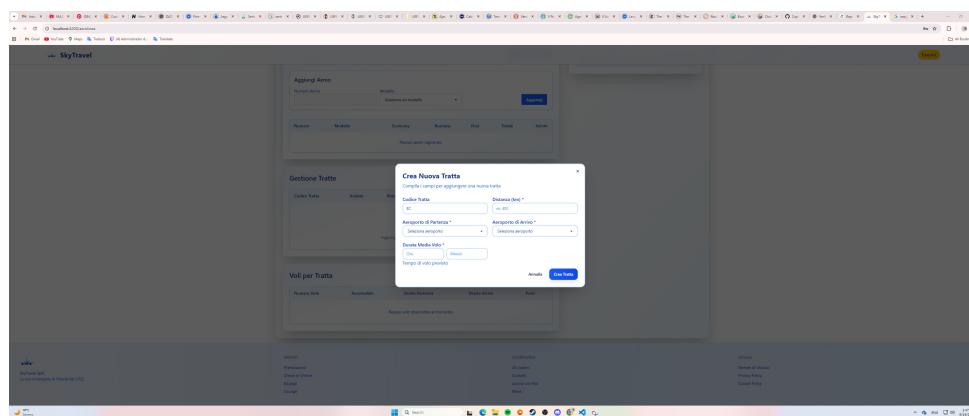


Figura 25: Form per l'aggiunta di una tratta

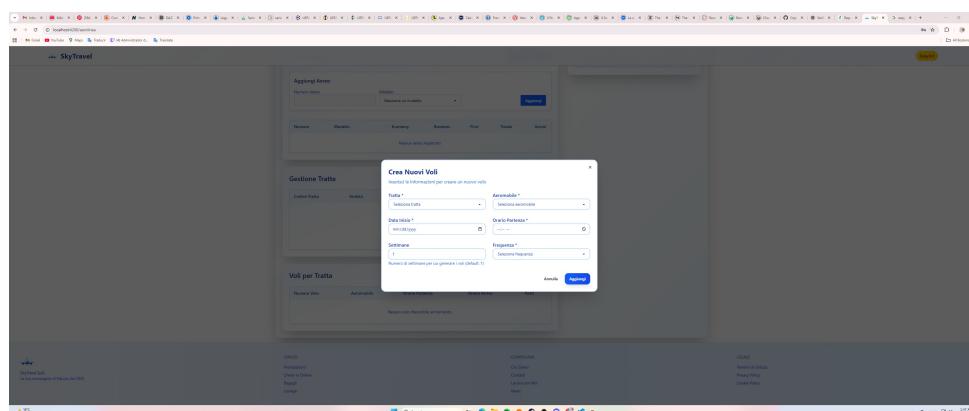


Figura 26: Form per l'aggiunta dei voli

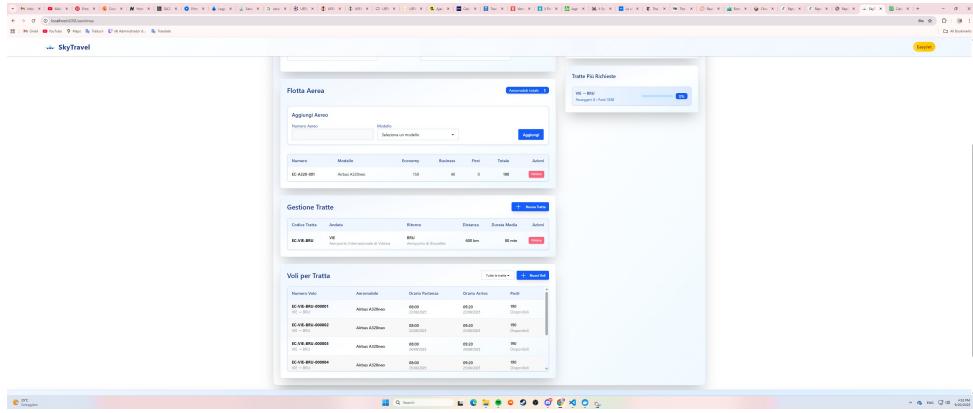


Figura 27: Esito nella creazione dell'aereo, della tratta e dei voli

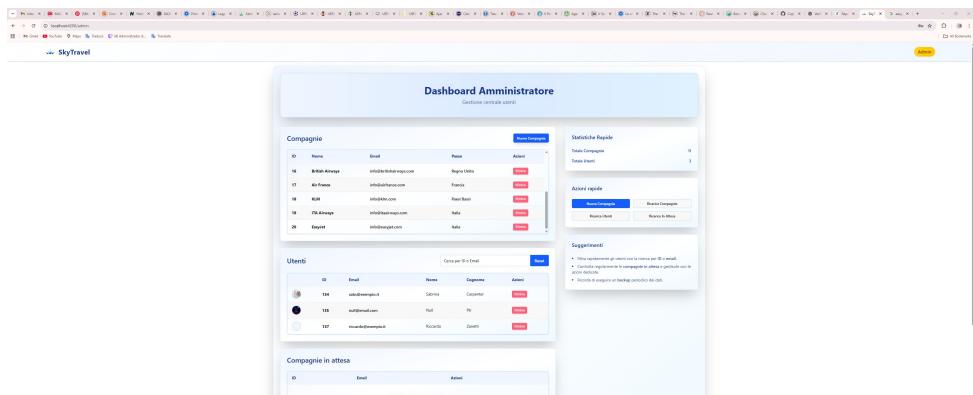


Figura 28: Conferma che la compagnia aerea ha effettuato l'accesso per la prima volta

## 7 Conclusioni

Il progetto implementa un ecosistema completo per la gestione end-to-end della prenotazione dei voli e del flusso aereo, con una netta separazione dei ruoli e un sistema di protezione basato su sessioni e JWT. È stato un lavoro che mi è sembrato davvero interessante e stimolante, è la prima volta in cui mi ritrovo nello sviluppo di una web application full stack. Ho avuto l'occasione di esplorare e mettere insieme frontend, backend e database, imparando moltissimo sia a livello tecnico sia di approccio progettuale. Sinceramente, considero questo progetto un'esperienza molto formativa e allo stesso tempo stra carina, poiché mi sono divertito tantissimo, anche se ci abbiamo messo un po'. Mi ha permesso di crescere e di acquisire nuove competenze con soddisfazione e ne sono contento.