Relazione tecnica sotourist

**Login**

L’utente può autenticarsi tramite le coppie (email, password). Nel caso di informazioni errate o mancanti, viene mostrato un messaggio di errore (con un alert personalizzato) e l’utente viene reindirizzato nuovamente alla pagina di login.

L’immagine mostra la schermata di login, definita in src/app/pages/login/.  
La parte grafica è realizzata in login.page.html, dove viene utilizzata una struttura ion-content con all'interno una card personalizzata e tre pulsanti (login, registrazione e accesso ospite). I campi ion-input sfruttano l’attributo label-floating per migliorare l’usabilità su mobile.

Il comportamento degli input è associato tramite ngModel alle variabili email e password, e viene gestita la pressione di ENTER per facilitare la navigazione (focusPassword(), triggerLogin()).

Il file login.page.ts definisce le principali funzionalità:

* **Caricamento dei dati inseriti dall’utente**.
* **Verifica della validità degli input**.
* **Chiamata al metodo saveProfile()**, che a sua volta invia la richiesta di login al backend.
* **Navigazione verso la pagina di registrazione (goToRegister())**.
* **Login ospite (loginAsGuest())**, con restrizioni e accesso temporaneo.

La validazione è di tipo **semplificato lato frontend**, poiché si disabilita il pulsante di login in caso di email o password mancanti. La validazione formale è demandata alle API del backend (POST /api/auth/login), che restituiscono anche lo stato utente aggiornato.

Il file login.page.scss definisce uno stile mobile-first coerente con la palette dell'applicazione:

* Bordeaux dominante per elementi di background e contrasto (#7B1E1E)
* Giallo senape (#FFCC00) per evidenziazioni
* Bianco come colore neutro di sfondo

**Registrazione**

La pagina di registrazione consente all’utente di creare un nuovo account fornendo **username, email e password**. In caso di dati mancanti o errati, viene mostrato un messaggio di errore (alert personalizzato) e il form non viene inviato.

La schermata è definita nella cartella src/app/pages/registrazione/.  
Il layout è strutturato in registrazione.page.html, dove un blocco centrale con sfondo dedicato contiene tre input (ion-input) con etichette flottanti (label-placement="floating"), per migliorare l’esperienza d’uso su mobile.

Il file registrazione.page.ts implementa:

* **Gestione dei dati inseriti** tramite [(ngModel)] per username, email, password.
* **Chiamata alla funzione onRegister()**, che invia i dati al backend tramite l’API POST /api/auth/register. In questa fase, l’utente è sempre registrato con tipo iniziale "standard".
* **Navigazione verso la pagina di login** con goToLogin().

Il file registrazione.page.scss utilizza uno stile coerente con il tema dell’app, con particolare attenzione alla leggibilità degli input e alla disposizione verticale centrata.

L’interfaccia è mobile-first, con uso esteso di componenti Ionic (come ion-input e ion-button), e si integra con un sistema di validazione lato backend che assicura l’univocità dell’email e la corretta formattazione della password.

**Home**

La pagina **Home** rappresenta il punto di ingresso principale per l’utente dopo l’autenticazione. Qui viene offerta una panoramica personalizzata basata sullo stato dei suoi viaggi, con la possibilità di crearne di nuovi o esplorare itinerari consigliati.

La vista è definita in src/app/home/ e utilizza il file home.page.html per la struttura dell’interfaccia.  
La pagina è composta dalle seguenti sezioni principali:

* **Header personalizzato**: Il componente <app-header> mostra il titolo “SoTourist”, un’icona e disabilita il pulsante "indietro".
* **Hero dinamico**: a seconda dello stato dell’utente:
  + Se **non ci sono viaggi attivi**, viene mostrata un’immagine promozionale con un pulsante per la creazione di un nuovo viaggio (openCreate()).
  + Se esiste un **viaggio attuale o futuro**, viene mostrata la città, una copertina dinamica e un pulsante per accedere all’itinerario (openItinerary()).
  + La versione "compatta" del hero appare al ridursi dello scroll, mantenendo le informazioni principali sempre visibili.
* **Destinazioni Trend**: una sezione che propone destinazioni popolari basate su immagini, con possibilità di iniziare direttamente un nuovo itinerario da una città suggerita.
* **Itinerari Consigliati**: elenco di itinerari già pronti, navigabili tramite card grafiche e descrittive.
* **Citazione finale**: una citazione motivazionale a completare la user experience.

La logica è definita in home.page.ts, dove vengono gestiti:

* I dati dinamici dell’utente (currentTrip, nextTrip).
* Le chiamate ai metodi openCreate(city?: string) e openItinerary(id: string) per la navigazione.
* Il comportamento del layout in base allo scroll (isShrunk, onHeroClick()...).

Lo stile (home.page.scss) segue la logica **mobile-first** e utilizza elementi visivi coerenti con il brand, come:

* **Immagini full-width**, overlay e blur.
* **Effetti dinamici con scroll e hover**, che migliorano la UX su dispositivi mobili.

**Creazione Itinerario**

La pagina di creazione guidata di un itinerario consente all’utente di inserire i parametri essenziali per costruire un viaggio personalizzato: città, alloggio e date. È strutturata come **un wizard a 4 step**, con passaggi sequenziali e riepilogo finale.

La pagina è definita in src/app/pages/crea/, e utilizza crea.page.html per l’interfaccia. Ogni step viene presentato come una “card” a larghezza piena, e l’intera sequenza è scorrevole orizzontalmente tramite trasformazioni CSS (translateX(...)).

I passaggi sono:

1. **Scelta della città**  
   Utilizza il componente custom <app-google-autocomplete>, con tipo ['(cities)'], integrato con le API di Google Maps. La validazione è effettuata sul campo selezionato.
2. **Scelta dell’alloggio**  
   L’utente può selezionare una struttura ricettiva (lodging) oppure un indirizzo specifico, con input dinamico e filtro tramite bounding box della città. Anche qui si usa <app-google-autocomplete>.
3. **Selezione del periodo**  
   Le date vengono selezionate tramite il componente <app-range-calendar-lite>, che permette una UI semplificata per mobile, con blocco delle date passate.
4. **Riepilogo e conferma**  
   Viene visualizzato un riepilogo delle informazioni inserite e l’utente può confermare tramite il pulsante “Conferma”, che attiva la funzione confirm() per creare l’itinerario.

Il file crea.page.ts gestisce:

* Il passaggio tra step (step, prevStep(), next-btn).
* Il salvataggio dei dati selezionati (setAccommodation(), setDates()).
* Il richiamo alla funzione finale confirm(), che invia le informazioni al backend (POST /api/users/:userId/itineraries), anche salvando temporaneamente in localStorage in caso di bozza.

Lo stile (crea.page.scss) è altamente responsive e ottimizzato per dispositivi mobili. I pulsanti seguono lo stile dell’applicazione con colori coerenti (bordeaux, giallo senape). Le card degli step sono ben distanziate per favorire la leggibilità e l’interazione touch.

**Viaggi**

La pagina “Viaggi” è il punto centrale in cui l’utente può visualizzare e gestire i propri itinerari, sia completati che in bozza. Viene mostrata subito dopo la creazione di un itinerario, ed è accessibile anche dalla barra di navigazione principale.

Il codice è definito in src/app/pages/viaggi/ e utilizza viaggi.page.html per la struttura. La pagina sfrutta diversi componenti personalizzati per organizzare le informazioni:

* <app-user-hero>: un'intestazione dinamica che mostra statistiche dell’utente loggato (numero viaggi, mete visitate, ecc.).
* <app-trip-card>: utilizzato per mostrare viaggi **in corso**, **imminenti** o **futuri**, con possibilità di accedere all’itinerario o eliminarlo.
* <app-unfinished-card>: visualizza gli **itinerari in bozza**, permettendo di riprendere o cancellare l’itinerario incompleto.

Il comportamento della pagina è dinamico: le sezioni vengono mostrate o nascoste in base alla presenza dei dati (inCorso, imminente, futuri, drafts) e allo stato dell’utente (isGuest).

Il file viaggi.page.ts gestisce:

* Il recupero dei dati tramite chiamate API (GET /api/users/:userId/itineraries?filter=...).
* La rimozione degli itinerari tramite deleteTrip() o deleteDraft().
* La navigazione verso la pagina di creazione (goToCreate()) o lo storico (openStorico()).

L’interfaccia include anche un **Floating Action Button (FAB)** doppio:

* Il lato sinistro consente di creare un nuovo viaggio (+).
* Il lato destro apre lo **storico dei viaggi passati** (🕒).

Il file SCSS (viaggi.page.scss) definisce uno stile modulare e adattivo, con enfasi sull’ordine visivo tra sezioni, e utilizza card-grid per l’impaginazione delle anteprime viaggio.

**Itinerario**

La pagina **Itinerario** consente all’utente di visualizzare e modificare le informazioni essenziali della propria bozza di viaggio prima della generazione completa. È accessibile dalla sezione "Viaggi", cliccando su un itinerario in stato di bozza o già generato.

La vista è definita nella cartella src/app/pages/itinerario/.  
L’interfaccia si articola in tre sezioni principali:

1. **Luoghi**
   * L’utente può selezionare luoghi imperdibili, ristoranti o luoghi già visitati.
   * Ogni categoria è rappresentata da una riga cliccabile (ion-item) che apre un **modale personalizzato** per la selezione.
   * I luoghi selezionati vengono mostrati tramite chip in preview.
   * L’inserimento utilizza <app-google-autocomplete>, con filtri specifici su tipo di luogo (mustSee, eat, visited) e bounding box limitato alla città selezionata.
2. **Preferenze**
   * Permette la scelta del mezzo di trasporto (walk, car, bus, bike), lo stile di vacanza (Standard, Shopping, Avventura, Relax) e un **prompt personalizzato per l’AI**.
   * Ogni preferenza apre un componente modale (in-page) dove l’utente può selezionare l’opzione desiderata.
3. **Generazione**
   * L’utente può cliccare su **“Genera Itinerario”**, che attiva la funzione generateItinerary().
   * Se l’itinerario è già stato generato o si è in attesa, il pulsante è disabilitato con messaggio dedicato.
   * L’avanzamento è mostrato tramite <app-generation-overlay> che mostra un messaggio di caricamento.

Il file itinerario.page.ts gestisce:

* Il recupero dei dati della bozza (trip, tripMustSee, ecc.)
* Le selezioni temporanee nei modali (modalTempPlaces)
* Le funzioni per aggiunta, rimozione, conferma (saveModal(), removeTempPlace(i))
* Il salvataggio delle preferenze e del prompt AI (selectValue(), tripPrompt)

Lo stile, in itinerario.page.scss, è progettato per una UX fluida e coerente su dispositivi mobili. Ogni sezione è separata graficamente, con header e icone distintive, e l’uso di chip e liste rende immediata la comprensione delle informazioni presenti.

**Panoramica Itinerario**

La pagina **Panoramica** mostra in modo sintetico la struttura dell’itinerario generato, organizzando visivamente le giornate pianificate con indicazioni su progressi e attività.

Questa schermata è accessibile subito dopo la generazione automatica dell’itinerario, e consente una prima consultazione prima di passare alla personalizzazione dettagliata.

Il codice si trova in src/app/pages/panoramica/, con struttura HTML in panoramica.page.html.

Le sezioni principali sono:

* **Hero dinamico**
  + Mostra un'immagine di copertina del viaggio (heroPhotoUrl), la città (getFormattedCity()), le date e l’alloggio selezionato.
  + Il background si adatta dinamicamente, e l’intestazione ha un overlay con opacità regolabile.
* **Lista giorni dell’itinerario**
  + Ogni giornata è rappresentata da una card (ion-card) con:
    - Titolo "Giorno X"
    - Attività sintetiche della giornata (getDayItems(i))
    - Data e numero attività
    - Barra di avanzamento suddivisa in segmenti: **mattina**, **pomeriggio**, **sera**, ognuno riempito in percentuale in base alla pianificazione (getStepProgress(...))
    - Pulsante “matita” per accedere alla personalizzazione dettagliata (vaiAPersonalizzazione())
* **Comportamento dinamico**
  + Se non ci sono giorni (daysCount === 0), viene mostrato un messaggio esplicativo.
  + I metodi openDay(i) e vaiAPersonalizzazione() consentono di navigare nella struttura interna delle giornate.

Lo stile (panoramica.page.scss) è ottimizzato per dispositivi mobili, con attenzione alla chiarezza visiva, all’uso dei colori tematici dell’app e alla progressività delle informazioni mostrate.

**Mappa**

La pagina **Mappa** visualizza su Google Maps le tappe di un giorno selezionato dell’itinerario, suddivise per **slot temporale** (mattina, pomeriggio, sera). È accessibile dalla sezione “Panoramica”, cliccando su una specifica giornata.

La struttura della pagina si trova in src/app/pages/map/, e si basa su map.page.html.

Le componenti principali sono:

* **Mappa interattiva**
  + Inizializzata nel contenitore #map, con utilizzo completo delle **Google Maps API**, inclusi marker personalizzati per le tappe.
  + La mappa è dinamicamente centrata e aggiornata in base alla giornata e alla fascia oraria selezionata.
* **Selettori**
  + **Selettore giorno**: mostra un elenco dei giorni disponibili per l’itinerario (days[]), con possibilità di navigazione tra i vari giorni (selectDay()).
  + **Selettore parte del giorno**: consente di filtrare le tappe per **mattina**, **pomeriggio** o **sera** (selectTimeSlot()).
* **Drawer tappabile con card**
  + Nella parte inferiore è presente un **drawer espandibile**, contenente una lista di <app-luogo-card> per ciascuna tappa della giornata (todayPlaces[]).
  + Le card mostrano nome, categoria e sono cliccabili per aprire il dettaglio.
* **Modale dettaglio**
  + Alla selezione di una card viene aperto un modale con:
    - Foto (se presente)
    - Nome
    - Indirizzo
    - Rating (se disponibile)
    - Pulsante per apertura diretta in Google Maps

Il file map.page.ts gestisce:

* La logica di caricamento delle tappe (refreshPlaces()).
* La gestione del drawer e della selezione card (openPlace(i)).
* L’inizializzazione della mappa con marker, bounds, clustering.
* La logica per cambiare giorno o slot orario.
* Il link diretto a Google Maps (openInGoogleMaps()).

Lo stile in map.page.scss è progettato per un utilizzo mobile-first:

* Il drawer può essere compatto o espanso, adattandosi al contesto.
* L’interfaccia è pulita, con spazio per l’interazione mappa + UI overlay.

**Panoramica Itinerario**

La pagina **Panoramica** mostra in modo sintetico la struttura dell’itinerario generato, organizzando visivamente le giornate pianificate con indicazioni su progressi e attività.

Questa schermata è accessibile subito dopo la generazione automatica dell’itinerario, e consente una prima consultazione prima di passare alla personalizzazione dettagliata.

Il codice si trova in src/app/pages/panoramica/, con struttura HTML in panoramica.page.html.

Le sezioni principali sono:

* **Hero dinamico**
  + Mostra un'immagine di copertina del viaggio (heroPhotoUrl), la città (getFormattedCity()), le date e l’alloggio selezionato.
  + Il background si adatta dinamicamente, e l’intestazione ha un overlay con opacità regolabile.
* **Lista giorni dell’itinerario**
  + Ogni giornata è rappresentata da una card (ion-card) con:
    - Titolo "Giorno X"
    - Attività sintetiche della giornata (getDayItems(i))
    - Data e numero attività
    - Barra di avanzamento suddivisa in segmenti: **mattina**, **pomeriggio**, **sera**, ognuno riempito in percentuale in base alla pianificazione (getStepProgress(...))
    - **Due modalità di interazione**:
      * Toccare il **card intero** apre la **pagina Mappa** per quel giorno (openDay(i)).
      * Toccare l’**icona matita** in alto a destra apre la **pagina di Personalizzazione** (vaiAPersonalizzazione()), permettendo di modificare le tappe di quella giornata.
* **Comportamento dinamico**
  + Se non ci sono giorni (daysCount === 0), viene mostrato un messaggio esplicativo.
  + I metodi openDay(i) e vaiAPersonalizzazione() consentono di navigare nella struttura interna delle giornate.

Lo stile (panoramica.page.scss) è ottimizzato per dispositivi mobili, con attenzione alla chiarezza visiva, all’uso dei colori tematici dell’app e alla progressività delle informazioni mostrate.

**Impostazioni**

La pagina **Impostazioni** consente all’utente di configurare preferenze personali legate all’interfaccia dell’app. È accessibile tramite il tab /tabs/settings.

Il codice si trova nella cartella src/app/pages/impostazioni/, e la struttura è definita nel file impostazioni.page.html.

La pagina è composta da una lista (ion-list) di tipo inset, organizzata in sezioni. Le funzionalità implementate sono:

**🔘 Tema**

* L’utente può attivare/disattivare la modalità **tema scuro/chiaro**.
* Il controllo è rappresentato da un componente <app-toggle> (custom), posizionato nella parte destra dell’ion-item.
* L’icona cambia dinamicamente tra 🌙 e ☀️ in base allo stato (darkMode).
* La funzione associata toggleDarkMode() aggiorna il tema dell’interfaccia, probabilmente modificando anche classi CSS globali o preferenze salvate (es. in localStorage).

**🌐 Lingua**

* L’utente può scegliere la lingua dell’app attraverso un ion-select con interfaccia popover.
* Attualmente è attiva solo l’opzione “Italiano”, mentre altre lingue (es. “English”) sono presenti nel codice ma **commentate**.
* Alla modifica della lingua viene eseguito il metodo changeLanguage(), che aggiorna la preferenza a livello di app (probabilmente integrata con un sistema i18n interno o esterno).

**UI e stile**

* La UI è semplice, mobile-first, e coerente con il resto dell’app.
* L’uso di icone (ion-icon) aiuta a identificare rapidamente ogni impostazione.
* Lo stile SCSS (impostazioni.page.scss) gestisce visivamente il toggle, evidenziando ad esempio lo stato attivo con la classe dark-mode-active.

**Modifica Date**

La pagina **Modifica Date** consente all’utente di aggiornare l’intervallo temporale del proprio itinerario, selezionando nuove date tramite un calendario interattivo. È una schermata di supporto che può essere aperta durante o dopo la creazione/modifica di un viaggio.

Il codice si trova in src/app/pages/modifica-date/ e la struttura è definita nel file modifica-date.page.html.

**Funzionalità**

* Mostra un riepilogo del viaggio attuale (nome città).
* Utilizza il componente <app-range-calendar-lite> per consentire la **selezione di un intervallo di date**.
* Il calendario impedisce la selezione di date precedenti al giorno corrente ([minDate]="today").
* Una volta selezionate le date, viene mostrato un riepilogo dinamico (newStartDate, newEndDate).
* Il pulsante “Salva e continua” è abilitato solo dopo che entrambe le date sono state selezionate.

**Logica**

* Il metodo onDatesSelected($event) riceve l’intervallo dal componente del calendario.
* Alla conferma, saveDates() aggiorna i dati del viaggio (presumibilmente tramite chiamata al backend PUT /api/users/:userId/itineraries o simile).
* Il file .ts gestisce il controllo iniziale della validità (\*ngIf="trip"), il caricamento dati e la logica di salvataggio.

**Stile**

Il file modifica-date.page.scss imposta:

* Una UI chiara, centrata verticalmente.
* Tipografia leggibile, spaziatura adeguata.
* Coerenza con il tema mobile-first del progetto.

**Cambio Credenziali**

**La pagina Cambio credenziali consente all’utente autenticato di modificare le proprie informazioni di accesso: email, username e password. È parte della sezione Impostazioni avanzate.**

**La pagina è definita in src/app/pages/change-password/, con struttura gestita nel file change-password.page.html.**

**📩 Modifica Email e Username**

* **Il primo blocco del form contiene due ion-input per modificare:**
  + **username**
  + **email**
* **I campi vengono inizializzati con i valori attuali (usernameP, emailP) e modificati tramite ngModel.**
* **Il pulsante “Salva Mail” è abilitato solo se i dati sono cambiati e validi.**
* **La funzione associata saveMail() invia i nuovi dati al backend (PUT /api/auth/users/:userId), aggiornando il profilo.**

**🔐 Cambio Password**

* **Il secondo blocco consente la modifica della password.**
* **Richiede tre campi:**
  + **Password attuale (password)**
  + **Nuova password (newPassword)**
  + **Conferma nuova password (confirmPassword)**
* **Tutti i campi sono obbligatori. Il pulsante “Salva Password” è disabilitato se mancano input.**
* **Alla conferma, viene invocato savePassword(), che invia la richiesta al backend per il cambio credenziali, probabilmente con verifica bcrypt lato server.**

**✅ Validazioni**

* **Le validazioni sono di tipo semplificato lato frontend (campi vuoti, confronto dati attuali).**
* **Si presume una validazione completa lato backend (conferma vecchia password, controllo sicurezza nuova password).**

**💡 UI e UX**

* **L'interfaccia è mobile-first, divisa in due blocchi visivi (form-block), con pulsanti distinti per email/username e password.**
* **Gli input usano label-placement="floating" per una UX pulita anche su schermi piccoli.**
* **Lo stile (change-password.page.scss) mantiene coerenza cromatica con il tema dell’applicazione, usando pulsanti color="success" per conferma azioni.**

**Profilo**

**La pagina Profilo consente all’utente di consultare e gestire le informazioni principali del proprio account, tra cui dati personali, abbonamento attivo, sicurezza e immagine profilo.  
È accessibile dalla sezione Impostazioni (/tabs/settings).**

**La struttura si trova in src/app/pages/profilo/, con markup definito in profilo.page.html.**

**📄 Informazioni Utente**

* **Viene visualizzato un riepilogo grafico del profilo:**
  + **Immagine utente tramite componente <app-profile-icon>, con supporto per modifica ([editable]="true").**
  + **Username e email.**
  + **Badge abbonamento dinamico (es. Standard, Premium, Gold), con classe CSS basata sul tipo (getBadgeClass()).**
  + **Data iscrizione (registrationDate).**

**🎖️ Gestione Abbonamento**

* **Sezione dedicata allo stato del piano attuale:**
  + **Visualizza il tipo (subscriptionPlan) e, se presente, la data di scadenza (subscriptionExpiry).**
  + **Il pulsante “Gestisci abbonamento” richiama la funzione goToUpgrade(), che porta alla pagina per cambiare o disdire il piano (/api/auth/users/:userId/upgrade / cancel).**

**🛡️ Sicurezza**

* **Sezione con azioni importanti legate alla sicurezza account:**
  + **Cambio password, tramite pulsante che apre la pagina Cambio credenziali.**
  + **Eliminazione account (permanente), con pulsante confirmDeleteAccount() che apre un modale di conferma. Se confermato, invia una richiesta DELETE /api/auth/users/:userId.**

**🎨 UI e Stile**

* **Tutti i blocchi sono impaginati in ion-card o ion-list con stile boxes-style, coerente con il resto dell’app.**
* **Gli elementi sono ottimizzati per il mobile e ben separati visivamente.**
* **Lo stile profilo.page.scss si occupa della formattazione del badge, delle dimensioni dell’immagine profilo, e della responsività generale.**
* **La classe delete-text evidenzia l’opzione “Elimina account” con colore distintivo (es. rosso).**

**Storico Viaggi**

**La pagina Storico Viaggi consente all’utente di visualizzare tutti gli itinerari passati, già conclusi, offrendo la possibilità di consultarli o eliminarli. È accessibile tramite il pulsante “orologio” nella sezione “Viaggi”.**

**Il codice della pagina è contenuto in src/app/pages/storico-viaggi/, con struttura gestita in storico-viaggi.page.html.**

**📜 Funzionalità principali**

* **Se la lista pastTrips[] contiene elementi, vengono mostrati tramite il componente <app-trip-card>, riutilizzato anche in altre sezioni dell’app.**
* **Ogni card di viaggio include:**
  + **Immagine copertina**
  + **Titolo del viaggio**
  + **Informazioni su durata e stile**
  + **Pulsante per aprire l’itinerario (goToTrip($event))**
  + **Pulsante per eliminare il viaggio (deleteTrip($event)), che rimuove definitivamente il record.**
* **Se non sono presenti viaggi passati, viene mostrato un messaggio informativo tramite ng-template:**

**“Non hai ancora viaggi passati.”**

**🧩 Composizione tecnica**

* **Il componente <app-navigation-bar> viene usato per impostare il titolo della pagina e fornire un tasto “indietro” verso /tabs/viaggi.**
* **L’interfaccia è interamente gestita tramite \*ngIf, \*ngFor e ng-template di Angular.**
* **Le azioni di navigazione e cancellazione sono gestite nel file .ts, che si occupa anche di recuperare i viaggi passati dal backend tramite chiamata GET /api/users/:userId/itineraries?filter=past.**

**🎨 UI e stile**

* **L’interfaccia è mobile-first, con ion-content e padding responsivo.**
* **I viaggi vengono visualizzati in un contenitore trip-list, impaginati verticalmente.**
* **Il messaggio di fallback è centrato e stilizzato con ion-text color="medium" per evidenziare lo stato informativo senza confondere con un errore.**

**Navigazione a Tab**

**L’applicazione utilizza una struttura a schede (tabs) per permettere una navigazione rapida e coerente tra le sezioni principali: Home, Viaggi, e Impostazioni. Questa struttura è definita nella cartella src/app/pages/tabs/, nei file tabs.page.ts, tabs.page.html, tabs.routes.ts.**

**📐 Struttura e Layout**

**La pagina tabs.page.html imposta un layout con:**

**html**

**CopiaModifica**

**<ion-tabs>**

**<ion-tab-bar slot="bottom"> ... </ion-tab-bar>**

**</ion-tabs>**

**All’interno di ion-tab-bar sono presenti tre ion-tab-button:**

1. **Home**
   * **Icona: home / home-outline**
   * **Rotta: /tabs/home**
   * **Mostra una pillola attiva (tab-pill) quando selezionato**
2. **Viaggi**
   * **Icona: airplane / airplane-outline**
   * **Rotta: /tabs/viaggi**
   * **Stile personalizzato per icona aereo (airplane-icon)**
3. **Impostazioni**
   * **Icona: settings / settings-outline**
   * **Rotta: /tabs/settings**

**🔄 Routing**

**Il file tabs.routes.ts definisce le rotte secondarie annidate sotto /tabs, mappando ciascuna scheda al rispettivo modulo:**

* **path: 'home' → modulo Home**
* **path: 'viaggi' → modulo Viaggi**
* **path: 'settings' → modulo Impostazioni**

**Questa configurazione consente un lazy loading efficiente e mantiene la barra inferiore fissa durante la navigazione tra le sezioni figlie.**

**🎨 Stile e UX**

* **La classe ion-selected viene applicata dinamicamente in base al valore di selectedTab, per evidenziare la tab attiva.**
* **Ogni tab mostra una pillola colorata (tab-pill) sopra l’icona quando selezionata, per rinforzare visivamente l’attivazione.**
* **I ion-label sono presenti nel codice ma commentati: si privilegia un design icon-only, più compatto e coerente con il focus mobile-first.**

**Questa struttura di navigazione fornisce una UX fluida, reattiva e chiara, con accesso rapido alle sezioni più importanti dell’app.**

**Upgrade Abbonamento**

**La pagina Upgrade consente all’utente di visualizzare e selezionare un piano di abbonamento tra quelli disponibili: Standard, Premium, e Gold. L’interfaccia offre una panoramica comparativa chiara e interattiva delle opzioni, con gestione diretta tramite backend.**

**La struttura si trova in src/app/pages/upgrade/, con contenuto visuale gestito da upgrade.page.html.**

**🎯 Funzionalità**

* **All’apertura, la pagina mostra una testata introduttiva con invito all’attivazione e nota sulla prova gratuita di 14 giorni.**
* **Utilizza un carousel Swiper (swiper-container, swiper-slide) per scorrere tra i piani disponibili:**
  + **Ogni piano ha:**
    - **Immagine illustrativa**
    - **Nome e prezzo**
    - **Elenco funzionalità**
    - **Pulsante di selezione**
* **I pulsanti per selezionare un piano invocano rispettivamente:**
  + **onStandard()**
  + **onPremium()**
  + **onGold()  
    Questi metodi effettuano la chiamata POST /api/auth/users/:userId/upgrade con { "plan": "premium" } (o gold), aggiornando il piano dell’utente.**

**🧾 Piani disponibili**

* **Standard**
  + **Gratis**
  + **Funzionalità base**
  + **Presenza di pubblicità**
  + **Personalizzazione limitata**
* **Premium – 4,99€/mese**
  + **Rimozione pubblicità**
  + **Accesso limitato all’AI**
  + **Personalizzazione avanzata**
* **Gold – 9,99€/mese**
  + **Tutto il Premium**
  + **Vantaggi aggiuntivi (es. status Gold, supporto, badge profilo, ecc.)**

**🖌️ Interfaccia e UX**

* **Lo Swiper mostra una card verticale per ogni piano, centrata e ottimizzata per dispositivi mobili.**
* **Le card sono stilizzate con classi dinamiche (standard, premium, gold), e l’utente corrente può visualizzare evidenziato il proprio piano attivo (currentType).**
* **La UI è pulita, coerente e informativa. I pulsanti sono visivamente differenziati per ogni piano.**