.DIEM

Università degli Studi di Salerno

Dipartimento di Ingegneria dell'Informazione
ed Elettrica e Matematica applicata (DIEM)

Documentazione di progetto

TravelNello

Corso di Mobile Programming – A.A. 2024/25

Gruppo 15:

Anthony Vita
Matricola: 0612707588

Simone Visconti
Matricola: 0612707795

Vincenzo Goffredo
Matricola: 0612708207

Antonio Radesca
Matricola: 0612708145

Indice

Introduzione e analisi preliminare

L’obiettivo del presente progetto è la realizzazione di un’app multipiattaforma per registrare i propri viaggi, con la possibilità di inserirne di nuovi, organizzare le varie destinazioni e consultare statistiche dettagliate sulle esperienze fatte.

Il design dell’app prevede 7 schermate principali:

- **Schermata principale**: visualizza l'elenco dei viaggi salvati, con un accesso rapido per aggiungere un nuovo viaggio, la possibilità di effettuare una ricerca tra i viaggi esistenti e la possibilità di applicare dei filtri di ricerca ai viaggi.
- **Schermata aggiunta nuovo viaggio**: consente l’inserimento di un nuovo viaggio con possibilità di specificare titolo, immagine, location, date di partenza e ritorno, descrizione e, opzionalmente, categorie.
- **Schermata dettaglio viaggio**: visualizza i dettagli relativi al viaggio selezionato, mostrando titolo del viaggio, immagine rappresentativa, località, date, note personali, categorie se presenti e descrizione. Consente inoltre di marcare il viaggio come preferito, fornendo inoltre l’accesso alla schermata di modifica del viaggio e al Trip diary.
- **Schermata modifica viaggio**: consente di modificare il viaggio selezionato, specificando nuovo titolo, località, date, descrizione, immagine e categorie.
- **Schermata Trip Diary**: è la schermata che mostra tutte note salvate all’interno del Trip Diary che stiamo visionando.
- **Schermata add Note**: consente di aggiungere note al trip Diary di un viaggio, definendo titolo, data, e immagini dalla galleria dello smartphone
- **Schermata dettagli singola nota Trip Diary**: consente di visualizzare la singola nota selezionata dal Trip Diary
- **Schermata modifica singola nota:** schermata che consente di modificare (o di eliminare) la singola nota che è stata selezionata
- **Schermata World Map**: visualizza una mappa del mondo con pin sulle località oggetto dei viaggi salvati

Considerazioni progettuali

Di seguito alcune prime considerazioni progettuali effettuate:

- **Approccio orientato al riuso e modularità**: la progettazione dell'applicazione ha visto un ampio uso di **components**, moduli riutilizzabili che vanno a comporre le schermate descritte.
- **Navigazione dinamica**: è stato usata la libreria **Stack Navigation** al fine di consentire un uso più agevole dell'applicazione
- **Modellizzazione dei dati**: sono state progettate e implementate classi secondo lo standard ECMAScript 2015 (**ES6**) al fine di garantire una migliore gestione dei dati, alleggerendo il carico computazionale.
- **App responsive**: Le dimensioni dei componenti e delle schermate vengono definite mediante l'oggetto **Dimensions** del package react-native. Attraverso tale oggetto è possibile ottenere le dimensioni effettive dello schermo del dispositivo su cui l'applicazione è in esecuzione, al fine di calcolare in modo dinamico le dimensioni dei singoli componenti e adattarli correttamente al display.
  Inoltre, l'interfaccia utente è progettata per adattarsi in modo efficace allo schermo sia in presenza di gesture che di tasti di navigazione tradizionali. Questo approccio consente di evitare la sovrapposizione tra elementi interattivi dell'interfaccia e le aree occupate dai sistemi di navigazione, garantendo così un'esperienza utente fluida e ottimizzata.

Components

1. navBar.js

Componente che implementa la barra di navigazione fissa in basso all'app.

- Visualizza tre icone: **Stats** (globo terrestre), **Home** (casa), **Add** (cerchio con +).
- Permette la navigazione rapida tra:
    - Schermata della mappa dei viaggi
    - Home con lista dei viaggi
    - Schermata di creazione nuovo viaggio
- Utilizza le **safe area insets** per adattarsi ai dispositivi con notch o barra inferiore.
- Ogni icona è un bottone touch che richiama la funzione di navigazione corrispondente.

---

2. tripCard.js

Componente grafico riutilizzabile che mostra un riassunto di ogni viaggio.

- Mostra un'immagine rappresentativa, titolo del viaggio, data di partenza e ritorno.
- L'intera card è cliccabile e apre i dettagli del viaggio.
- È possibile inserire immagini locali o da URL remoto.

3. categoryBox.js

Mostra tutte le categorie di viaggio disponibili sotto forma di box selezionabili.

- Ogni categoria ha icona e colore distintivo.
- Supporto a selezione multipla.
- Evidenzia graficamente le categorie selezionate.
- Mostra messaggio informativo se nessuna categoria è selezionata.

---

4. customHeader.js

Header personalizzato per la schermata principale.

- Sfondo a **gradiente azzurro/blu**.
- Titolo **“TravelNello”** con font decorativo.

5. noTripsAlert.js

Card/alert che appare quando la lista dei viaggi è vuota.

- Presenza di un sottotesto che invita alla creazione di un viaggio.
- L’intera card è cliccabile e apre la schermata di creazione viaggio.

---

6. searchBar.js

Barra di ricerca avanzata per filtrare i viaggi nella home

- Input testuale per ricerca per titolo.
- Icona filtro che apre il `FilterModal`.
- Mostra un **badge rosso** sull’icona filtro se sono attivi dei filtri.

7. filterModal.js

Modale impiegato per l’applicazione dei filtri alla lista viaggi

- Solo preferiti
- Categorie multiple
- Date (partenza, ritorno, intervallo)
- Interfaccia grafica intuitiva.
- Pulsanti per **applica**, **cancella**, **azzera**.
- Validazione delle date integrata.

---

8. button.js

Componente bottone riutilizzabile

- Azioni principali come **salvataggio**, **aggiornamento**, **eliminazione**.
- Accetta testo, colore e funzione onClick.

9. datePickerInput.js

Input per la selezione delle date (partenza, ritorno con aggiunta di note)

- Apre il date picker nativo (Android/iOS).
- Gestione della formattazione e data minima selezionabile.

---

10. descriptionEditor.js

Editor testuale esteso per le descrizioni (viaggi o note)

- Textarea multilinea.
- Toolbar per formattazione base Markdown (grassetto, corsivo, bullet).
- Conteggio caratteri in tempo reale.
- Mostra errore se si supera il limite caratteri.

---

11. locationInput.js

Input con suggerimenti automatici di località tramite API **OpenCage**

- Mostra fino a 4 suggerimenti in base all'input dell'utente.
- Selezionando un suggerimento, il campo si riempie automaticamente.
- Include placeholder, icona posizione, gestione cancellazione suggerimenti.

12. imageSearchPicker.js

Permette la selezione di immagini da **Pexels** o da immagini locali

- Mostra anteprima dell'immagine scelta.
- Fallback su immagini locali se query breve o errore.
- Immagini locali precaricate per funzionamento offline.

---

13. noteCard.js

Piccola card per mostrare una nota del diario

- Titolo, data, anteprima del testo (senza markdown).
- Anteprima della prima immagine (se presente).
- Badge con numero immagini presenti.
- Cliccando apre il dettaglio della nota.

---

14. phoneImagePicker.js

Permette di selezionare immagini dalla galleria del telefono

- Mostra le immagini in griglia.
- Possibilità di rimuovere immagini singolarmente.
- Gestione dei permessi con alert in caso di negazione.

15. inputBox.js

Input di testo riutilizzabile in vari form

- Supporta input singolo o multilinea.
- Include **label** e **placeholder**.
- Gestione padding, bordo e allineamento coerente con il design dell’app.

Models and Data

1. Models

Trip (models/TripClass.js)

Rappresenta un viaggio con le sue proprietà fondamentali: id, titolo, immagine, date di partenza/ritorno, location, descrizione, categorie (come stringa di nomi delle categorie separati da virgola), e stato "favorite". Espone anche il metodo toggleFavorite() per gestire i preferiti.

TripDiary (models/TripDiary.js)

Ogni viaggio ha un diario associato che contiene una lista di note (notes, oggetti DiaryNote). Gestisce anche il prossimo id disponibile per una nuova nota (nextNoteId). Espone metodi per aggiungere, aggiornare, rimuovere e recuperare note.

DiaryNote (models/DiaryNote.js)

Ogni nota ha un id univoco, un riferimento al diario, titolo, data, contenuto (markdown) e un array di immagini.

Categories (models/categories.js)

Le categorie sono definite come oggetti con nome, icona e colore.

TripCollector (models/TripCollector.js)

La classe TripCollector gestisce tutti i viaggi e i diari.
Mantiene due mappe (Map) in memoria: una per i viaggi e una per i diari.
Espone metodi per:

- aggiungere, aggiornare, rimuovere viaggi e note
- recuperare viaggi, diari, note singole o tutte
- creare automaticamente un diario associato ad ogni nuovo viaggio
- mantenere i contatori progressivi per id di viaggi e note

Persistenza con Async Storage

Per la persistenza dei dati viene usata la libreria **AsyncStorage** e vengono implementati dei metodi e funzioni ad hoc che gestiscono il salvataggio ed il caricamento dei dati:

Salvataggio:

Ogni volta che viene modificato un viaggio o un diario, TripCollector serializza i dati (in JSON) e li salva su AsyncStorage tramite le chiavi trip_collector_data (viaggi) e trip_diaries_data (diari).

Caricamento:

All'avvio dell'app, nel file App.js viene eseguito initializeTripCollector() che ricostruisce oggetti e associazioni in memoria leggendo i dati da AsyncStorage.

2. Data

tripsDataManagment (data/tripsDataManagment.js)

Crea ed esporta una sola istanza di TripCollector chiamata 'tripCollectorA'.

Questa istanza viene importata ovunque servano dati, garantendo che tutta l'app lavori sempre sugli stessi dati in memoria e su disco.

Screens

HomeScreen

La schermata principale dell'app funge da hub centrale per la gestione e la visualizzazione dei viaggi dell'utente. Il componente recupera dinamicamente i dati tramite il metodo `tripCollectorA.getAllTrips()`, garantendo la sincronizzazione in tempo reale con lo stato globale dell'applicazione e la persistenza dei dati tramite AsyncStorage.

La SearchBar implementa un filtro testuale in tempo reale, aggiornando lo stato dei risultati a ogni input dell'utente. Un'icona filtro apre una modale (FilterModal) che permette la selezione di filtri avanzati (categorie, viaggi preferiti, intervalli di date). Tali filtri sono gestiti tramite uno stato centralizzato che, insieme al search, viene passato a un sistema di filtraggio che opera su array immutabili, garantendo la consistenza e l'atomicità delle operazioni.

Ogni viaggio viene visualizzato tramite il componente TripCard che riceve in props i dati essenziali (`title`, `image`, `departureDate`, `returnDate`, ecc.) e funzioni di callback per la navigazione ai dettagli. Se non esistono viaggi, viene mostrata una card (NoTripsAlert) che invita l'utente a creare un nuovo viaggio.

In fondo, una NavBar consente la navigazione rapida verso le schermate principali, mantenendo la separazione delle responsabilità tra navigazione e logica della schermata attiva.

La schermata sfrutta i React Hooks (`useState`, `useEffect`, `useCallback`) per la gestione della UI reattiva e per garantire aggiornamenti efficienti e coerenti in risposta a mutamenti dello stato globale e dei filtri.

CreateTripScreen

Questa schermata funge da form per la creazione di un nuovo viaggio, orchestrando più componenti controllati e stateless. Il campo titolo implementa validazione immediata (max 20 caratteri, obbligatorio) con feedback visivo tramite stato. L'immagine può essere selezionata tramite ImageSearchPicker, che offre sia la ricerca su API Pexels sia la selezione da immagini locali già incluse nel bundle dell'app (fallback robusto).

La località utilizza un componente (LocationInput) con suggerimenti automatici asincroni tramite interrogazione all'API OpenCage, favorendo la precisione e la standardizzazione dei dati. Le date di partenza e ritorno sono gestite con un

DatePicker nativo che, oltre alla UX fluida, previene l'inserimento di dati non validi tramite props come `minimumDate`.

La descrizione è gestita tramite DescriptionEditor, che implementa funzioni di markdown live-editing, con toolbar dinamica e validazione della lunghezza (max 3000 caratteri). Le categorie sono selezionabili tramite CategoryBox, che supporta selezione multipla e un feedback visuale su base colore/icona.

Alla pressione del pulsante di salvataggio, la validazione globale del form è eseguita in modo centralizzato; in caso di successo, viene creato un oggetto Trip con id auto-incrementale (`tripCollectorA.getNextId()`), e salvato persistentemente tramite `tripCollectorA.addTrip()`. Al termine, l'utente viene reindirizzato alla HomeScreen tramite navigation stack, propagando un refresh per la lista.

TripDetailsScreen

Questa schermata mostra in modo dettagliato tutte le informazioni di un viaggio selezionato, recuperate tramite `tripCollectorA.getTrip(tripId)`. L'immagine di copertina è presentata in full-width, con overlay trasparente per titolo e località. Le date sono evidenziate tramite icone e layout verticale, migliorando la leggibilità cronologica.

Le categorie sono visualizzate come chip colorati con icone (basate su MaterialCommunityIcons), generate dinamicamente dal campo `category` del viaggio. La descrizione viene renderizzata in markdown tramite il componente `react-native-markdown-display`, supportando formattazione avanzata e una UX ricca.

L'utente può interagire tramite icone: la stella dei preferiti chiama `trip.toggleFavorite()` e persiste il cambiamento con `tripCollectorA.updateTrip()`, mentre l'icona edit consente l'accesso alla ModifyTripScreen. Un pulsante centrale consente di accedere al TripDiaryScreen, mantenendo la coerenza della navigazione contestuale.

La schermata sfrutta `useLayoutEffect` per la personalizzazione dell'header della navigation bar e per la gestione degli eventi di navigazione (swipe, back button).

ModifyTripScreen

Schermata dedicata all'editing di un viaggio esistente, con campi precompilati tramite props e stato locale inizializzato dal viaggio selezionato (`tripCollectorA.getTrip(tripId)`). Il form replica la struttura di CreateTripScreen ma ogni campo è inizializzato e controllato, e ogni modifica viene aggiornata in tempo reale.

La validazione dei dati segue le stesse regole della creazione, garantendo consistenza e prevenendo errori di input. Due pulsanti principali: uno salva le modifiche tramite `tripCollectorA.updateTrip()`, l'altro elimina il viaggio con conferma tramite alert (`tripCollectorA.removeTrip()`).

In caso di eliminazione o modifica, viene effettuato un redirect verso la HomeScreen, forzando il refresh della lista. Se il viaggio non viene trovato, viene mostrato un messaggio di errore e la schermata si blocca in uno stato di sola lettura.

StatsScreen

Questa schermata offre una rappresentazione visuale delle località visitate tramite un componente MapView (`react-native-maps`). Tutte le località dei viaggi sono aggregate e deduplicate, quindi geocodificate tramite chiamate asincrone all'API OpenCage. Ogni marker rappresenta una località visitata, con badge in overlay che mostra il numero totale di località. L'intero flusso di caricamento è gestito tramite stato (`loading`), con ActivityIndicator durante le chiamate di rete, garantendo trasparenza e feedback all'utente. La schermata è completamente responsive e si adatta dinamicamente alle dimensioni dello schermo.

TripDiaryScreen

Raccoglie e visualizza tutte le note associate a un viaggio tramite FlatList, ordinandole per data decrescente. I dati sono recuperati tramite `tripCollectorA.getDiaryForTrip(tripId)`, garantendo isolamento tra i diari dei diversi viaggi.

Ogni nota appare tramite NoteCard, che espone titolo, data, breve anteprima e, se disponibili, thumbnail delle immagini. Un header statico con titolo dinamico e icona libro introduce l'area. In assenza di note, viene visualizzato un messaggio informativo con icona, migliorando l'onboarding dell'utente.

Un floating action button consente la creazione di nuove note, portando l'utente alla DiaryNoteEditScreen in modalità "creazione". La navigazione sfrutta la propagazione di parametri (`tripId`, `noteId`) per garantire l'accesso diretto ai dettagli o alla modifica di una nota.

DiaryNoteDetailsScreen

Mostra i dettagli di una singola nota di diario: titolo, data formattata in modo descrittivo, contenuto renderizzato tramite markdown e una galleria di immagini. I dati vengono recuperati dal diario associato via `tripCollectorA.getDiaryForTrip(tripId).getNote(noteId)`.

Il layout è ottimizzato per la lettura: il markdown permette la formattazione avanzata del testo, mentre la galleria immagini visualizza una preview ordinata delle foto associate. Un'icona nell'header consente l'accesso diretto alla modifica della nota. La schermata gestisce in modo robusto il caso di nota non trovata, mostrando un messaggio di errore.

DiaryNoteEditScreen

Permette la creazione o la modifica di una nota di diario, gestendo in modo lo stato dei campi: titolo (max 50 caratteri, obbligatorio), data (gestita tramite DatePickerInput), contenuto (editor markdown con supporto toolbar), e immagini (caricabili tramite il selettore PhoneImagePicker che interagisce con la media library del dispositivo).

La validazione dei campi è immediata e i messaggi di errore sono visibili in tempo reale. Alla conferma, la nota viene salvata tramite `tripCollectorA.addNoteToDiary()` o aggiornata tramite `tripCollectorA.updateNote()`, con persistenza in AsyncStorage. Il pulsante elimina è disponibile solo in modalità modifica e chiede conferma prima della cancellazione (`tripCollectorA.removeNote()`).

Dopo ogni operazione, l'utente viene riportato al TripDiaryScreen con la lista aggiornata, garantendo la coerenza tra UI e stato dati.

App.js

App.js è il file principale di Travelnello: qui avviene il caricamento dei dati salvati in precedenti sessioni, il settaggio dello stack navigator, ed altre operazioni fondamentali per il corretto funzionamento dell'applicazione.

Le varie sezioni che compongono il file vengono di seguito descritte:

Importazione delle dipendenze e dei componenti principali

All'inizio del file vengono importati tutti i moduli necessari per la gestione della navigazione, della grafica, dello stato globale e delle risorse dell'app. In particolare:

- Viene utilizzato **@react-navigation/native** e **@react-navigation/stack** per la gestione della navigazione a stack tra le schermate.

- Sono importati i componenti React Native di base (`View`, `ActivityIndicator`, `Text`, `StyleSheet`) e il provider **SafeAreaProvider** per la corretta visualizzazione su dispositivi con notch o barre di sistema.

- Con **expo-asset** e `useFonts` vengono caricati in modo asincrono le immagini locali e il google font usato nell'header.

- Vengono importate tutte le schermate (HomeScreen, TripDetailsScreen, CreateTripScreen, ecc.) che compongono il flusso dell'applicazione.

- Si importa l'istanza di `tripCollectorA`, che rappresenta la sorgente dati persistente per tutti i viaggi e i diari dell'utente.

Caricamento delle risorse e dei dati persistenti

Questa sezione è dedicata al caricamento iniziale delle risorse essenziali prima di mostrare l'interfaccia all'utente:

1. Si utilizza lo stato `isLoading`, inizialmente impostato a `true`, per mostrare una schermata di caricamento durante la fase iniziale.
2. La funzione `preloadImages` sfrutta `Asset.loadAsync` per precaricare le immagini locali di fallback, garantendo che siano immediatamente disponibili.
3. Nel blocco `useEffect`, viene definita una funzione asincrona che:
    - Precarica le immagini locali.
    - Inizializza il `tripCollectorA` caricando tutti i dati persistenti da **AsyncStorage** (viaggi e diari).
    - Applica un breve delay per assicurare la sincronia completa tra caricamento dati e UI.
4. Viene effettuato anche il caricamento del font personalizzato tramite l'hook `useFonts`.
5. Se il caricamento non è ancora terminato (`isLoading` o `!fontsLoaded`), viene mostrata una schermata di loading con spinner e testo.

Configurazione della navigazione tra le schermate

Dopo il caricamento iniziale, viene costruito il navigation stack dell'app:

- Tutto viene wrappato all'interno di **SafeAreaProvider** e **NavigationContainer** per garantire la corretta gestione delle safe area e della navigazione.

- Si definisce uno stack di schermate tramite `Stack.Navigator`, impostando le opzioni grafiche comuni come colore, font e allineamento dell'header.

- Ogni schermata (HomeScreen, TripDetailsScreen, CreateTripScreen, ecc.) viene inserita nello stack con le rispettive opzioni:

    - La home ha l'header nascosto per usare un header custom.

    - Le altre schermate hanno titoli specifici e, dove necessario, header customizzati o parametri che modificano dinamicamente il titolo.

    - Alcuni screen (come la Home) possono ricevere parametri di refresh per forzare la ricarica dei dati dopo operazioni di creazione/modifica/eliminazione.

Stili per la schermata di caricamento

Alla fine del file è presente uno `StyleSheet` che definisce lo stile della schermata di caricamento:

- Il container centra il contenuto sia verticalmente che orizzontalmente e applica uno sfondo neutro.

- Il testo di caricamento è stilizzato per essere ben visibile e leggibile.