Relazione Gruppo 8

Ronca Ciro - Quaranta Davide - Barberio Gregorio

Indice

1) Introduzione

Il progetto realizzato è un'applicazione che permette la gestione della propria libreria personale dando la possibilità, all'utente, di aggiungere ed eliminare libri, ricercarli, categorizzarli e visualizzare l'intera collezione.

Il progetto è stato realizzato in React Native e, all'interno di questa relazione vengono mostrate alcune strategie di progetto utilizzate.

---

2) FileStorage

fileStorage.js ci da la possibilità di salvare, caricare ed eliminare i libri in modo persistente sulla memoria del dispositivo, grazie al file locale salvato nella memoria interna del dispositivo. Questo approccio consente di mantenere salvati i dati anche dopo la chiusura dell'app, garantendo una migliore esperienza all'utente. Questa classe è così composta:

```
import * as FileSystem from 'expo-file-system';
```

Questa riga importa tutte le funzionalità di **expo-file-system**, che permette di leggere, scrivere, creare o eliminare file nella memoria del dispositivo.

```
const FILE_URI = FileSystem.documentDirectory + 'libri.json';
```

Viene creata una constante che contiene il percorso completo del file **libri.json.**

- **FileSystem.documentDirectory** è un percorso che viene assegnato solo all'app stessa, dove può salvare file persistenti.
- **libri.json** è il nome del file che conterrà i dati dei libri, ed è formato in questo modo:

```
[
 {
  "id": '',
  "title": '',
  "author": '',
  "type": '',
  "status": '',
  "rating": '',
  "notes": '',
 }
]
```

---

3) UseFocusEffect

Per garantire il corretto funzionamento dell'applicazione, è stato necessario implementare una funzione in grado di ricaricare dinamicamente la lista dei libri ogni volta che la schermata viene visualizzata.

Questa funzionalità è fondamentale sia per lo sviluppatore, che ha così la possibilità di gestire e aggiornare i dati in maniera dinamica all'interno delle varie schermate, sia per l'utente, che può così visualizzare in tempo reale i libri già presenti o appena aggiunti.

Abbiamo deciso di utilizzare l'hook **useFocusEffect**:

Questa riga serve a importare **useFocuesEffect** per poi utilizzarla.

```
import { useFocusEffect } from '@react-navigation/native';
```

```
useFocusEffect(
  useCallback(() => {
    const loadData = async () => {
      const data = await caricaLibri();
      setLibri(data);
    };
    loadData();
  }, [])
);
```

La funzione **loadData** è dichiarata come asincrona **(async)** perché il caricamento dei dati dalla sorgente esterna (**libri.json**) non è immediato e può richiedere del tempo. Abbiamo dovuto utilizzare la funzione **await** perché è necessario attendere che tutti i dati vengano completamente caricati, così da garantire che quelli utilizzati in altre funzioni non siano incompleti o non aggiornati.

---

4) TabNavigator

Una parte fondamentale della creazione del progetto è quella di poter far navigare l'utente tra le varie pagine. Per garantire un'esperienza utente semplice e intuitiva, abbiamo deciso di implementare una **Tab Navigator** posizionata nella parte inferiore dello schermo. Un esempio di implementazione della tab Categorie è:

```
<Tab.Screen name="Categorie" component={Categories}
options={{tabBarIcon: ({ focused }) => (
    <Image
      source={require('../../assets/tabIcon/catIcon.png')}
      style={{ width: 30,height: 30,}}/>
```

```
  ) }}/>
```

Questa è come la Tab si mostra una volta avviata la nostra applicazione.

Inoltre, abbiamo deciso che la prima schermata mostrata all'apertura dell'app fosse la Homepage. Questo è stato possibile impostando la pagina di homepage come
Initial Route:

Prima di tutto abbiamo importato il componente

```
import Homescreen from '../homescreen.js';
```

Poi lo abbiamo indicato come schermata iniziale

```
<Tab.Navigator initialRouteName="Homepage">
```

---

5) StackNavigator

Abbiamo utilizzato lo **Stack Navigator** per gestire la navigazione tra le schermate. Quando l'utente naviga verso una nuova pagina (ad esempio cliccando sulla schermata di Dettaglio o di Aggiunta), questa viene aggiunta allo stack di navigazione, permettendo poi di tornare indietro facilmente.

```
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import AppTabs from './src/navigation/AppTabs.js';
import BookDetails from './src/bookDetails.js';
import EditBook from './src/editBook.js';
import CategoryDetails from './src/categoryDetails.js';
import AddBook from './src/addBook.js';
import BookList from './src/allBooks.js';
import Categories from './src/categories.js';
```

Questo blocco di codice rappresenta la configurazione della navigazione dell'app.

- **NavigationContainer** gestisce lo stato di navigazione dell'app
- **createNativeStackNavigator** è una funzione che restituisce un oggetto che permette di definire il Navigator e i Screen.
- **AppTabs** rappresenta la navigazione tramite tab nella parte inferiore dell'applicazione.
- **import BookDetails from './src/bookDetails.js';** Questa riga come le successive, permette di registrare ogni componente all'interno dello Stack Navigator, consentendo poi la navigazione tra le varie schermate.

```
export default function App() {
 return (
  <NavigationContainer>
   <Stack.Navigator>
       <Stack.Screen name="Tabs" component={AppTabs}
        options={{ headerShown: false }} />
       <Stack.Screen name="Dettaglio" component={BookDetails} />
       <Stack.Screen name="Modifica Libro" component={EditBook} />
       <Stack.Screen name="Dettaglio Categoria"
        component={CategoryDetails} />
       <Stack.Screen name="Aggiungi Libro" component={AddBook} />
       <Stack.Screen name="La mia libreria" component={BookList} />
       <Stack.Screen name="Categorie" component={Categories} />
   </Stack.Navigator>
  </NavigationContainer>
 );
}
```

Il componente **NavigationContainer** deve trovarsi agli estremi perché è il contenitore principale che gestisce tutto il sistema di navigazione dell'applicazione.

---

6) ImagePicker

Nel nostro progetto abbiamo implementato una funzionalità che permette all'utente di selezionare un'immagine, dalla propria galleria, per impostare la copertina del libro. Ciò è reso possibile grazie a **ImagePicker** all'interno della funzione **pickImage.**

ImagePicker viene così importato

```
import * as ImagePicker from 'expo-image-picker';
```

E cosi viene implementato:

```
const pickImage = async () => {
  try {
    let result = await ImagePicker.launchImageLibraryAsync({
      mediaTypes: 'images',
      allowsEditing: true,
      aspect: [2, 3],
      quality: 1,
    });

    if (!result.canceled) {
      setImageUri(result.assets[0].uri);
      onImagePicked(result.assets[0].uri);
    }
  } catch (error) {
    console.error('Errore nella selezione dell\'immagine:', error);
  }
};
```

Abbiamo dovuto specificare diverse cose:

- La funzione è asincrona **async** per permettere l'apertura della galleria del dispositivo.
- **mediaTypes: 'images'** è fondamentale perché diamo la possibilità di scegliere solo le immagini.

- **setImageUri(result.assets[0].uri);** Salva il percorso URI dell'immagine per associare l'immagine al libro

```
return (
 <View style={{ alignItems: 'center', justifyContent: 'center', padding: 20}}>
  {!imageUri && (<TouchableOpacity onPress={pickImage}>
  <Image source={require('../assets/addImage.png')}/></TouchableOpacity>
  {imageUri && (<TouchableOpacity onPress={pickImage}>
   <Image
    source={{ uri: imageUri }}
    style={{ width: 100, height: 100, marginTop: 20, borderRadius: 10 }}
   /></TouchableOpacity> )} </View> );
```

Quando noi clicchiamo sull'immagine a sinistra si apre la nostra galleria. Questa blocco di codice ci permette di catturare l'immagine selezionata dalla nostra galleria in modo da poterla impostare come copertina. Ovviamente, se viene scelta un'immagine e si vuole cambiare, cliccando su quest'ultima viene data questa possibilità.

6a) Default.jpg

Viene data anche la possibilità di non inserire una foto, in quel caso viene impostata una di default.

```
<Image source={book.img ? { uri: book.img  }
  : require('../assets/default.jpg') }
  style={styleBookDetail.coverImage} />
```

In questa sezione di codice, presente in quasi tutte le schermate, viene eseguito un controllo che verifica se è stata inserita una foto. In caso contrario, viene utilizzata l'immagine di default a destra.

6b) Richiesta Accesso Galleria

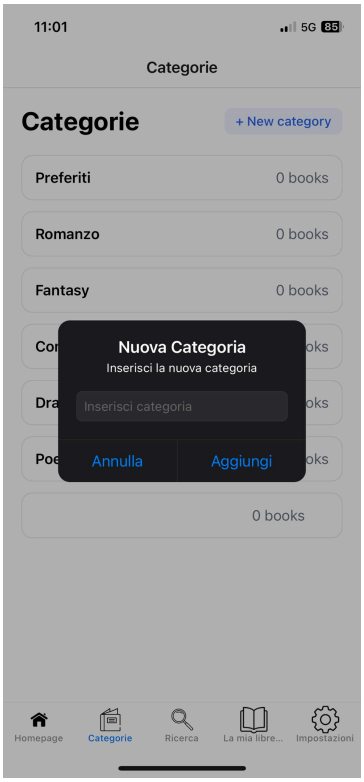
Abbiamo implementato una funzione che permette di richiedere l'accesso alla galleria del dispositivo. Questa è così strutturata

```
useEffect(() => {
  (async () => {
     const permission  = await ImagePicker.requestMediaLibraryPermissionsAsy
     if (permission.status !== 'granted') {
       Alert.alert('Permesso negato',
       'È necessario il permesso per accedere alla galleria.');
     }

  })();

}, []);
```

requestMediaLibraryPermissionAsync è il metodo per richiedere i permessi **.** Il risultato viene salvato in **permission.** Dopodichè se lo stato del permesso non è "granted" (cioè non è stato concesso), viene mostrato un avviso **(Alert)** all'utente.

7) DialogContainer

Nella schermata della Categoria viene data la possibilità di inserire una nuova categoria. Cliccando sul tasto “+ New category” compare un pop-up:



Ciò è reso possibile grazie a **Dialog Container:**

```
<Dialog.Container visible={visible}>

 <Dialog.Title>Nuova Categoria</Dialog.Title>
 <Dialog.Description>Inserisci la nuova categoria
 </Dialog.Description>
   <Dialog.Input
         placeholder="Inserisci categoria"
         value={newGenere}
         onChangeText={setNewGenere}
      />
<Dialog.Button label="Annulla"
onPress={handleCancel}/>
<Dialog.Button label="Aggiungi"
onPress={handleAdd}/>
</Dialog.Container>
```

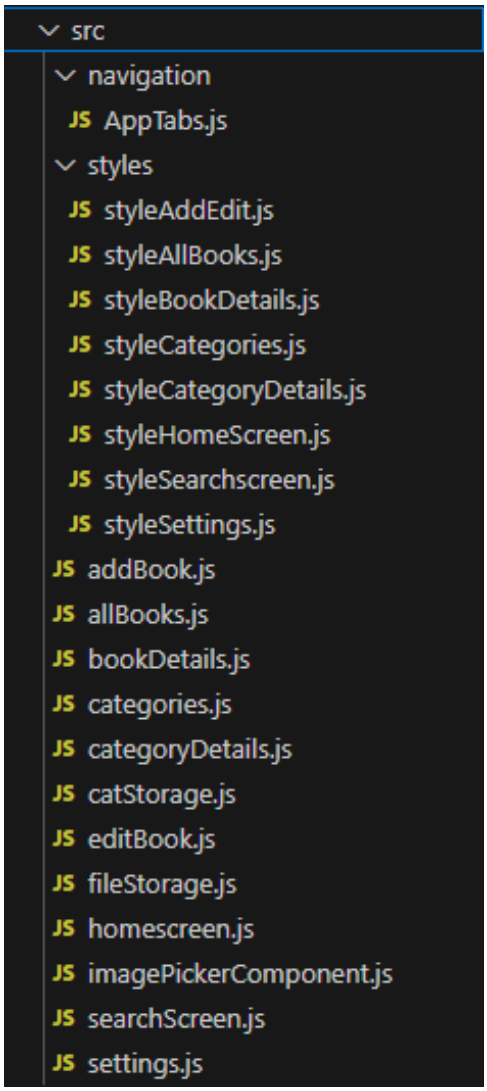
Quando l’utente scrive un nuovo genere e preme il tasto "Aggiungi", questo viene automaticamente visualizzato nella schermata delle categorie.

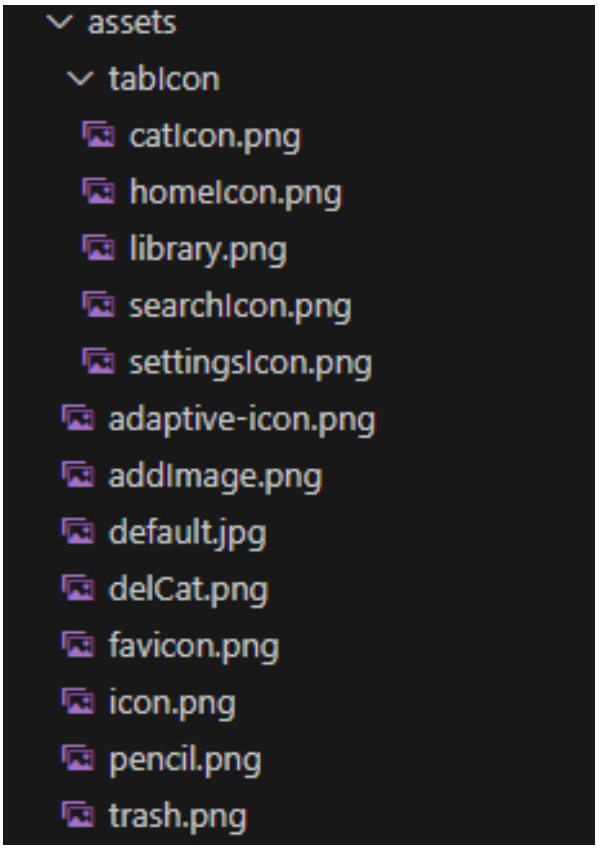
8) Organizzazione del Progetto

Un'altra strategia adottata si basa sull'organizzazione del progetto, ovvero quella di suddividere i file in base alla loro funzione all'interno dell'applicazione.

Abbiamo strutturato il progetto nel seguente modo:

- **src/**: contiene i file principali dell'applicazione
- **src/styles/**: raccoglie i file di stile utilizzati dalle diverse schermate, per mantenere separata la logica dalla presentazione grafica.
- **src/navigation/**: include la logica di navigazione, come la barra di navigazione che consente di spostarsi tra le varie pagine dell'app.





- **assets/**: raccoglie tutte le immagini presenti all'interno delle scena
- **assets/tabIcon**: raccoglie tutte le immagini usate dentro la Tab Navigation