Sport Mate

Relazione progetto LAM

Giovanni Spadaccini

giovanni.spadaccini3@studio.unibo.it

Indice

1 Introduzione

Sport Mate è un'applicazione progettata per organizzare e partecipare ad eventi sportivi nella propria zona.

L'obiettivo principale è quello di facilitare la connessione tra appassionati di sport, permettendo loro di creare, trovare e partecipare a varie attività sportive.

2 Scelta delle tecnologie

Dart è un buon linguaggio per diverse ragioni. È stato progettato per essere semplice e produttivo, con una sintassi chiara e coerente che facilita la scrittura e la manutenzione del codice. Inoltre, Dart offre prestazioni elevate grazie alla sua compilazione ahead-of-time (AOT) che consente di produrre codice nativo efficiente.

Flutter è preferibile a React Native per alcuni motivi chiave. Prima di tutto, Flutter offre una maggiore coerenza e controllo sull'aspetto e il comportamento delle applicazioni, grazie al suo set completo di widget personalizzabili. Inoltre, Flutter utilizza il motore grafico Skia, che garantisce prestazioni elevate e un rendering fluido su tutte le piattaforme. Infine, Flutter ha un supporto eccellente per lo sviluppo multi-piattaforma, permettendo di scrivere una sola volta il codice e distribuirlo su iOS, Android, web e desktop.

Per quanto riguarda il backend è stato scritto in python con fastApi, e un database postgrest.

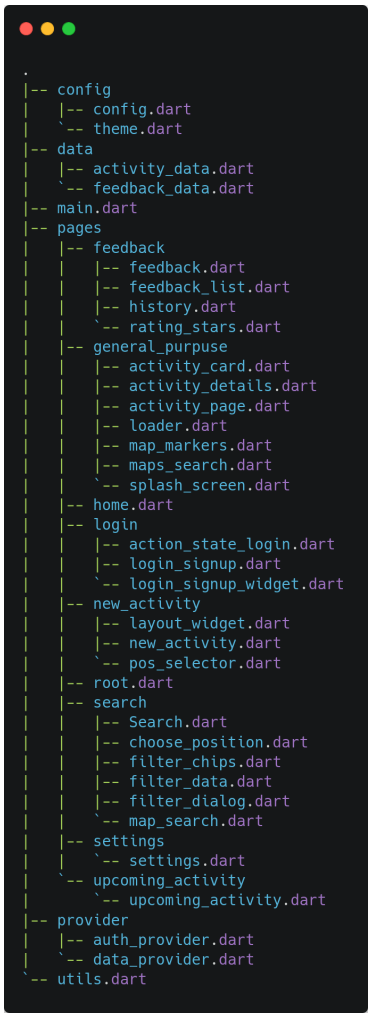
3 Features dell'applicazione

L'applicazione comprende diverse pagine chiave:

- **Login**: Pagina di accesso per autenticare gli utenti.
- **Signup**: Pagina di accesso creare un profilo utente.
- **Search**: Motore di ricerca per trovare eventi o partner sportivi nella tua zona, con possibilità d'impostare dei filtri. Supporta due modalità di visualizzazione:
  - **Mappa**: visualizza gli eventi con attraverso la loro posizione nella geografica
  - **Lista**: lista che permette di visualizzare con più dettagli gli eventi nella zona selezionata
- **Creazione Attività**: Interfaccia per creare nuovi eventi sportivi.
- **Storico**: Raccolta degli eventi passati a cui si è partecipato con la possibilità di aggiungere un feedback.
- **Ricordo**: Sezione per lasciare un messaggio di ricordo dell'attività, e aggiungerci un punteggio su quanto è stata gradita.
- **Visualizzazione di un'Attività e Partecipazione**: Pagina dettagliata di ogni evento, che include opzioni per iscriversi all'evento, ottenere informazioni aggiuntive e aggiungere una notifica per ricordarsi dell'evento.

4 Struttura del codice sorgente

La struttura del codice sorgente è divisa nelle seguenti cartelle:



- **config**: Contiene le configurazioni globali dell'applicazione che non cambiano durante l'esecuzione, o che se cambiano non hanno un'influenza sulla UI
- **data**: Contiene le strutture dati delle activity e del feedback.
- **provider**: Contiene i dati che, quando cambiano, hanno un riscontro nella UI e quindi dev'essere aggiornata anche la UI.
- **pages**: Sono definiti tutti i widget delle varie pagine e le pagine stesse.
- **utils.dart**: Contiene utility functions utilizzate in varie parti dell'applicazione.

I dati vengono caricati da un backend facendo richieste progressive (che si aggiornano rispetto all'ultima richiesta fatta) e vengono salvati in locale, così da averli offline.

5 Scelte implementative

5.1 Dati

Le due strutture di dati principali dell'applicazione sono definite sotto **data** e sono:

```
class Activity {
  final String description;
  final DateTime time;
  final LatLng position;
  final Attributes attributes;
  final int numberOfPeople;
  final int id;
  final List<String> participants;
  final String creator;
}

class Attributes {
  final String level;
  final int price;
  final String sport;
}
```

```
class FeedbackActivity {
  String username;
  int activityId;
  int rating;
  String comment;
}
```

- **Activity** 6: contiene i dati relativi ad un evento sportivo, come il nome, la descrizione, la posizione, la data e l'ora, il numero di partecipanti, ecc.
- **Feedback** 6': contiene i dati relativi al feedback lasciato da un utente su un'attività, come il punteggio, il messaggio, ecc.

5.2 Provider

Un provider in Flutter è una classe che gestisce e fornisce dati e stato all'applicazione, facilitando la separazione tra logica di business e interfaccia utente. Questo approccio migliora la leggibilità e la manutenzione del codice. Il provider viene inserito nel widget tree e consente a ogni widget di accedere e reagire agli aggiornamenti dello stato. Utilizzando il provider, si può garantire una gestione efficiente e reattiva dei dati, poiché i widget si aggiornano automaticamente quando il provider notifica un cambiamento di stato.

```dart
class DataProvider with ChangeNotifier {
  final storage = const FlutterSecureStorage();

  bool loading = false;
  bool isConnected = false;
  List<Activity> activities = [];
  List<FeedbackActivity> feedbacks = [];
  DateTime? lastUpdate = null;
  LatLng lastPos = LatLng(44.498955, 11.327591);
  bool loadingPos = false;

  Future<void> deleteAllStoredData() async {...}
  Future<void> loadFromStorage() async {...}
  Future<void> saveToStorage(DateTime lastUpdate) async {...}

  Future<void> load(token) async {
    await loadFromStorage();
    try {
      var lastUpdate = DateTime.now().subtract(lastUpdate.timeZoneOffset);
      this.feedbacks  = await loadFeedback(token);
      update_activity(await _loadActivitys(),await _deletedActivities(token))
      this.lastUpdate = DateTime.now();
      await saveToStorage(lastUpdate);
      loading = false; isConnected = true;
    } catch (e) {
      loading = false; isConnected = false;
    }
  }

  Future<List<FeedbackActivity>> loadFeedback(token) async {
    final req = await http.get(Uri.https(Config().host, '/feedback'),
        headers: {'Authorization': 'Bearer ${token}'});
    if (req.statusCode != 200)  throw Exception('Impossibile caricare i feedback');
    return json.decode(req.body)cast<FeedbackActivity>();
  }

  Future<List<Activity>> _loadActivitys(List<int> ids) async {
    Map<String, dynamic> params = {};
    if (lastUpdate != null) params['last_update'] = lastUpdate!.toIso8601String();
    return await http.get(Uri.https(Config().host, '/activities/search', params))
      .cast<Activity>();
  }

  Future<List<int>> _deletedActivities(token) async {...}
  void addFeedback(FeedbackActivity feedback) {..}
  void joinActivity(int id, String user) {...}
  void leaveActivity(int id, String user) {...}
  void deleteActivity(int id) {...}
}
```

Nell'applicazione come visto nella sezione di struttura del codice sorgente ne abbiamo due:

- **Activity Data**: specifica i metodi e i dati per caricare, modificare ed eliminare i dati relativi alle Activity 6
- **Auth**: specifica i metodi e i dati per gestire l'autenticazione e l'accesso dell'utente.

Activity Data in particolare è un provider che si occupa di caricare i dati delle activity, e di salvarli in locale, così da averli offline. Implementa la politica Single Source of Truth, ovvero che i dati vengono caricati da un backend facendo richieste progressive (che si aggiornano rispetto all'ultima richiesta fatta) e vengono salvati in locale, così da averli offline.

Auth questo provider permette di aggiornare tutta la UI quando l'utente si autentica o si disconnette.

5.3 Pagine

5.3.1 Home

All'esecuzione carica il file home, che controlla se è stato salvato un authetication token valido, se sì carica la pagina Search 5.5, se no carica la pagina di Registrazione e Login 5.4. Il cambio è fatto attraverso provider che, in base al cambiamento del token sceglie quale pagine caricare.

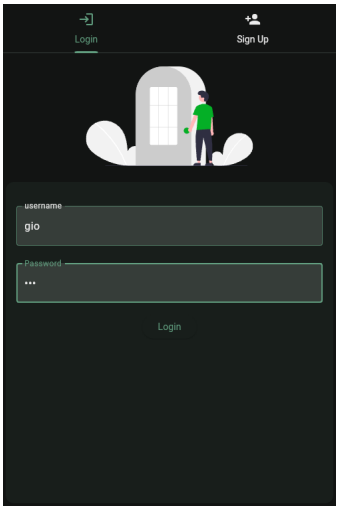
Quindi il codice della homepage è qualcosa come:

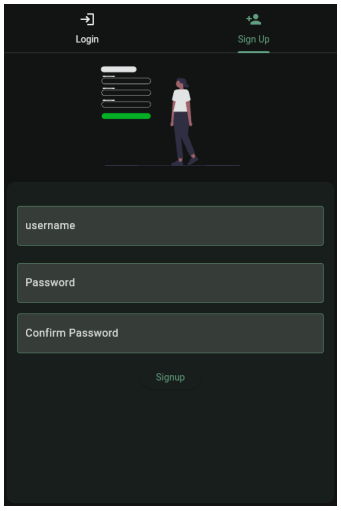
```
class Home extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider(create: (ctx) => AuthProvider()..tryAutoLogin()),
        ChangeNotifierProvider(create: (ctx) => DataProvider())
      ],
      child: MaterialApp(
        home: Consumer<AuthProvider>(builder: (ctx, auth, _) {
          return auth.loading
              ? CircularProgressIndicator()
              : auth.isAuthenticated
                  ? SearchPage()
                  : LoginSignupPage();
        }),
      ),
    );
  }
}
```

Inoltre nella home andiamo a definire anche il DataProvider che utilizzeremo dopo.

5.4 Registrazione e Accesso

Le prime pagine che si incontrano sono quelle di accesso e registrazione, queste pagine si connettono alle api e impostano l'authetication token attraverso l'auth provider, questo fa aggiornare la home page cambiando la pagina home a SearchPage 5.5.



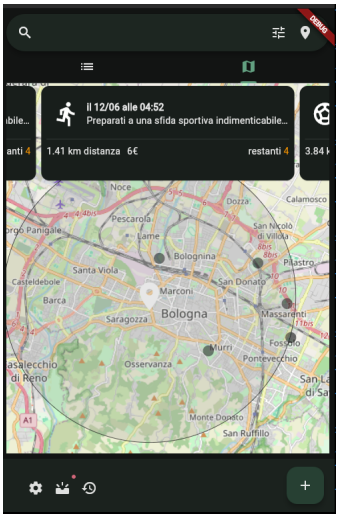


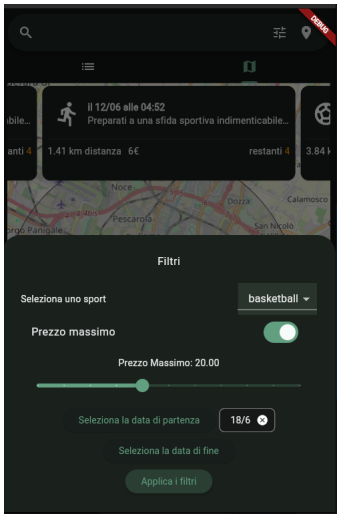
5.5 Search

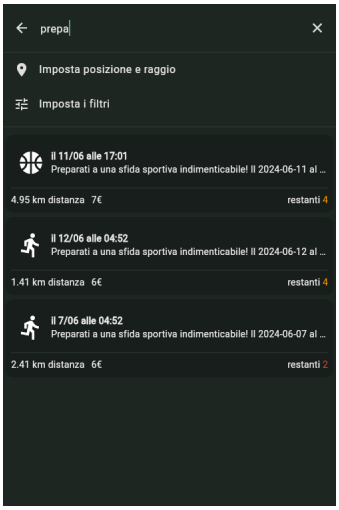
Search è la pagina più complessa dell'applicazione, permette di cercare eventi sportivi nella propria zona, con possibilità d'impostare dei filtri. Supporta due modalità di visualizzazione: Vediamo il codice ad alto livello, è composto da due widget stateful, il primo **SearchPage** che è un wrapper, carica i dati da DataProvider e aggiorna **_SearchPage** che contiene l'interfaccia della search. **SearchPage** è il widget root della pagine non autenticate, questo significa che quando si modificano dei dati significativi in Data Provider, questo ha aggiorna tutti i componenti sottostanti, cosa che nel nostro caso è accettabile anzi. La **_SearchPage** contiene altri dati come i filtri, la posizione attuale, e il raggio di ricerca.

```dart
class SearchPage extends StatefulWidget {
  @override
  State<SearchPage> createState() => _SearchPageState();
}

class _SearchPageState extends State<SearchPage> {
  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) {
      final authProvider = Provider.of<AuthProvider>(context, listen: false);
      Provider.of<DataProvider>(context, listen: false).load(authProvider.token!);
    });
  }

  @override
  Widget build(BuildContext context) {
    return Consumer<DataProvider>(builder: (context, dataProvider, child) {
      return SearchPage(key: UniqueKey(), data: dataProvider.toApplicationData());});
  }
}
```

```dart
class _SearchPage extends StatefulWidget {
  final ApplicationData data;
  _SearchPage({super.key, required this.data});
  @override
  State<_SearchPage> createState() => _SearchPageStateFilter(data);
}

class SearchPageStateFilter extends State<_SearchPage> {
  LatLng? pos;
  double radius = 5000;
  List<Activity> displayActivities = [];
  final SearchController searchController = SearchController();
  FilterData filterData = FilterData.init();
}
```

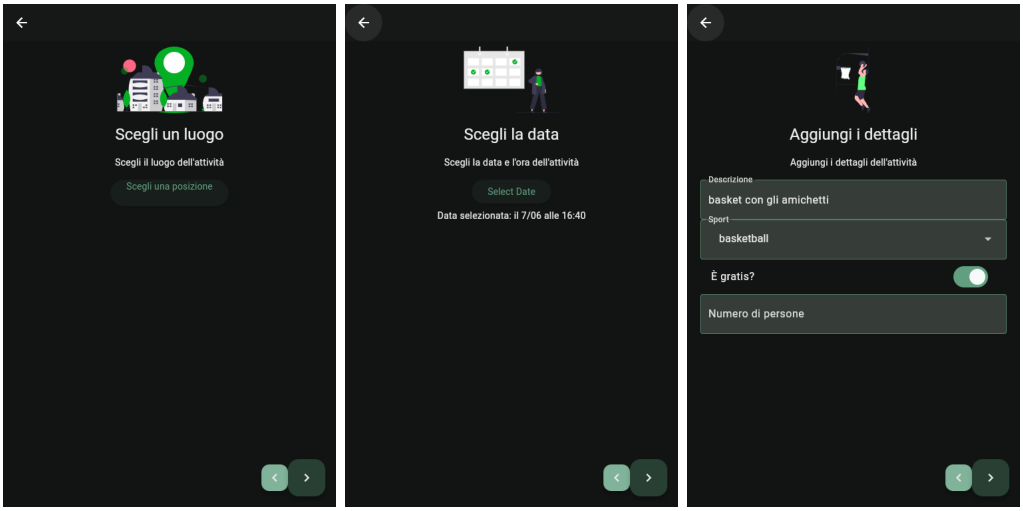






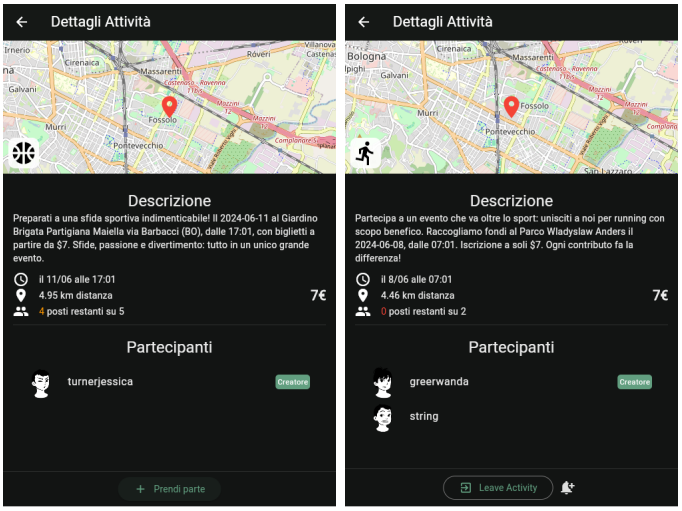
5.6 Aggiungi attività

La creazione nel creare un attività sportiva è molto semplice, si compone di un form che permette di inserire i dati dell'attività, e di un bottone per confermare l'aggiunta.



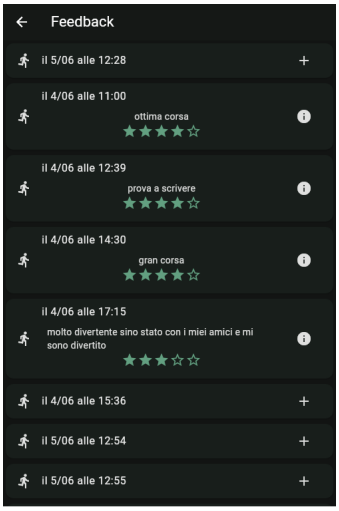
5.7 Partecipa ad un'attività

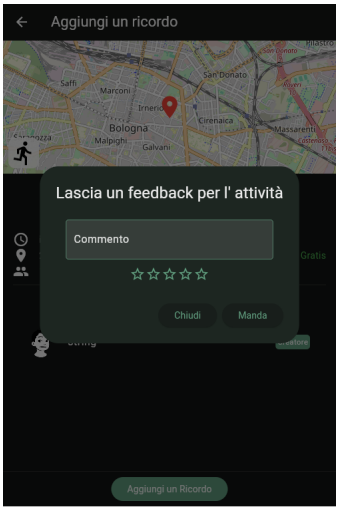
Per partecipare ad un'attività basta cliccare sul bottone partecipa, e si verrà aggiunti alla lista dei partecipanti. Inoltre si può aggiungere un promemoria per l'evento.



5.8 Storia

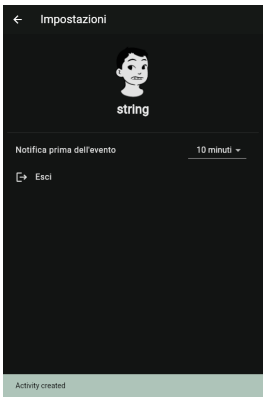
Permette di visualizzare gli eventi passati a cui si è partecipato, e di aggiungere un feedback. Inoltre mostra il feedback in quelli in cui lo si è già lasciato. Per aggiungere il feedback, va ad aggiornare sia il feedback locale attraverso il DataProvider, sia quello sul server.


Feedback
il 5/06 alle 12:28
il 4/06 alle 11:00
ottima corsa
il 4/06 alle 12:39
prova a scrivere
il 4/06 alle 14:30
gran corsa
il 4/06 alle 17:15
molto divertente sino stato con i miei amici e mi sono divertito
il 4/06 alle 15:36
il 5/06 alle 12:54
il 5/06 alle 12:55


Aggiungi un ricordo
Lascia un feedback per l' attività
Commento
Chiudi
Manda
Aggiungi un Ricordo

5.9 Impostazioni

Le impostazioni, mostrano informazioni sull'utente, e permettono di modificare le impostazioni che per ora è solo quanto prima dell'evento ricevere una notifica. Inoltre permette di disconnettersi. Le configurazioni a differenza di altre impostazioni non devono essere inserite dentro un Provider in quanto non influenzano la UI.



6 Glossario

- **Activity o Attività**: si riferisce ad un evento, un attività sportiva organizzata da un utente.
- **Feedback o Ricordo**: si riferisce alla creazione di un feedback o ricordo legata ad un'attività passata.