

Sport Mate

Relazione progetto LAM

Giovanni Spadaccini

giovanni.spadaccini3@studio.unibo.it



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.

Scelta delle tecnologie

Dart

- Semplice e produttivo
- Sintassi chiara e coerente
- Null safety

Flutter

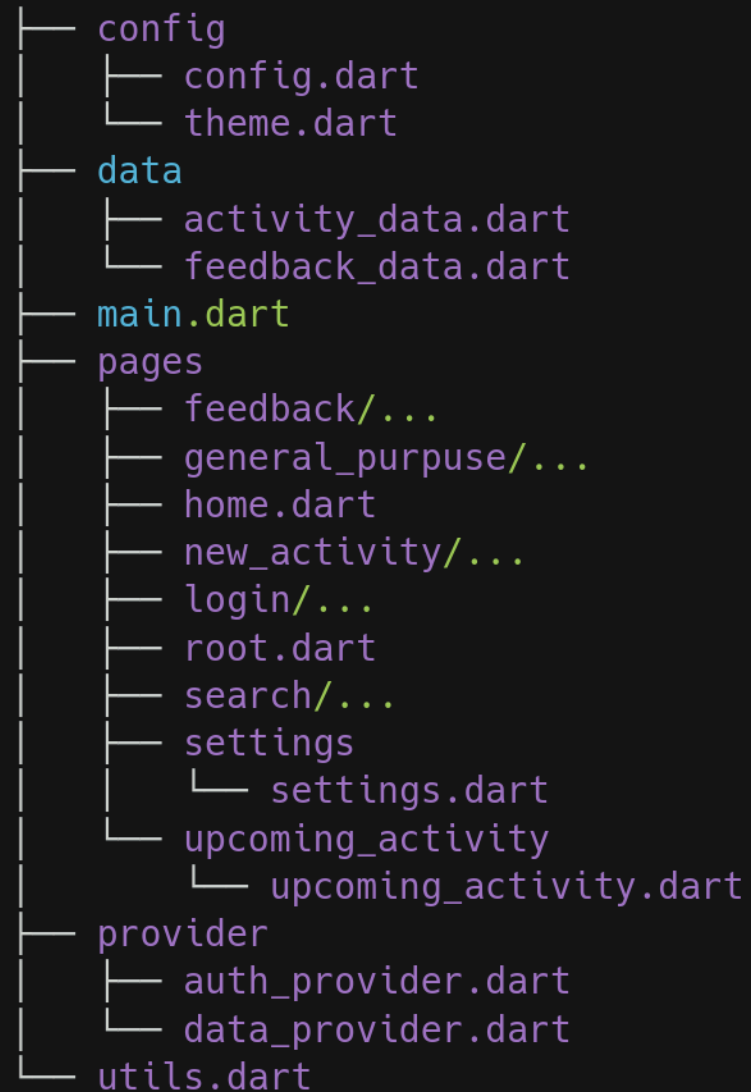
- Supporto eccellente per lo sviluppo multi-piattaforma, componeti definiti in maniera dichiarativa
- principali librerie utilizzate: provider, flutter_secure_storage, flutter_map, flutter_local_notifications

Backend

- Scritto in Python con FastAPI
- Database PostgreSQL

Features dell'applicazione

- **Login:** Autenticazione utenti
- **Signup:** Creazione profilo utente
- **Search:** Motore di ricerca per eventi o partner sportivi con filtri
 - **Mappa:** Visualizzazione eventi sulla mappa
 - **Lista:** Visualizzazione dettagliata degli eventi
- **Creazione Attività:** Creazione nuovi eventi sportivi
- **Storico:** Raccolta eventi passati con possibilità di aggiungere feedback
- **Ricordo:** Lasciare messaggi e punteggi degli eventi
- **Visualizzazione di un'Attività e Partecipazione:** Dettagli evento e iscrizione



```
— config
  — config.dart
  — theme.dart
— data
  — activity_data.dart
  — feedback_data.dart
— main.dart
— pages
  — feedback/...
  — general_purpose/...
  — home.dart
  — new_activity/...
  — login/...
  — root.dart
  — search/...
  — settings
    — settings.dart
  — upcoming_activity
    — upcoming_activity.dart
— provider
  — auth_provider.dart
  — data_provider.dart
— utils.dart
```

Struttura del codice sorgente

- **config:** Configurazioni globali
- **data:** Strutture dati delle attività e dei feedback
- **provider:** Dati che influenzano la UI
- **pages:** Widget delle varie pagine
- **utils:** Funzioni utilitarie

Dati caricati da un backend e salvati in locale per accesso offline.

```
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;  
}
```

Activity Data

Activity: Contiene dati relativi agli eventi sportivi (nome, descrizione, posizione, data, ora, partecipanti, ecc.)



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

FeedbackData

Feedback: Contiene dati
relativi ai feedback
(punteggio, messaggio, attività
associata)

```

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) {...}
}

```

Provider

Un provider gestisce e fornisce dati e stato all'applicazione, migliorando la separazione tra logica di business e interfaccia utente.

- **Activity Data:** Metodi e dati per caricare, modificare ed eliminare attività
- **Auth:** Metodi e dati per gestire autenticazione e accesso utente

Pagine

Andiamo a definire le pagine che compongono l'applicazione

Home

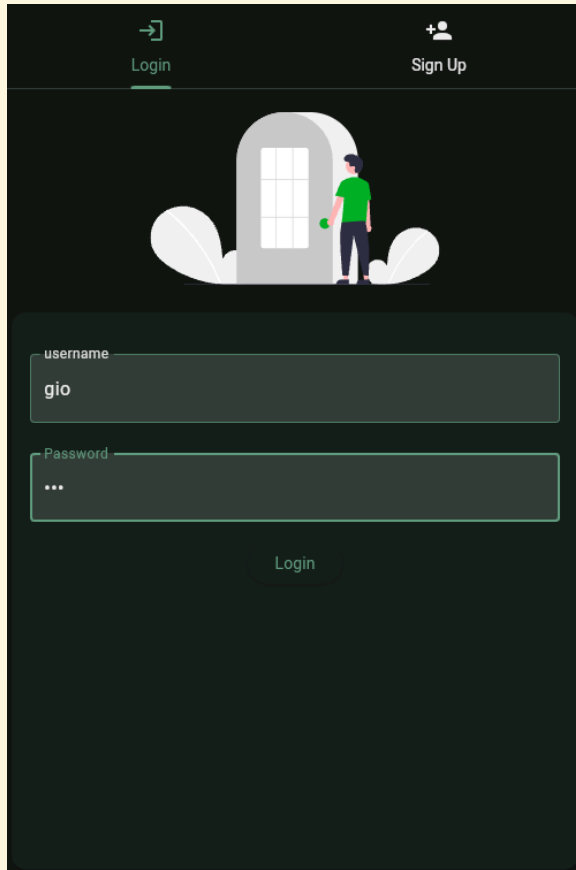
Carica il file home, che controlla se è stato salvato un authentication token valido.

- Carica la pagina **Search** se il token è valido
- Carica la pagina di **Registrazione e Login** se il token non è valido

```
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();
        })),
    );
  }
}
```

Registrazione e Accesso

Prima pagina visualizzata per accesso e registrazione.
Questa utilizza il provider Auth per registrare il token.



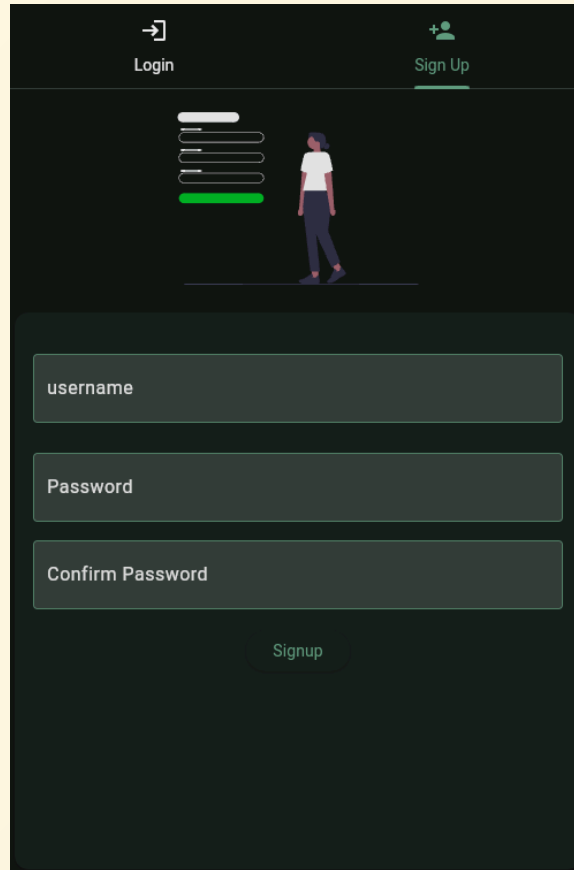
The login screen features a dark background with a top navigation bar containing 'Login' and 'Sign Up' links. The 'Login' link is underlined. Below the navigation bar is an illustration of a person in a green shirt standing in front of a white door. The main form area includes a 'username' label above a text input field containing 'gio', and a 'Password' label above a password input field with three dots. A 'Login' button is positioned at the bottom of the form.

Login Sign Up

username
gio

Password
...

Login



The sign up screen has a dark background with a top navigation bar containing 'Login' and 'Sign Up' links. The 'Sign Up' link is underlined. Below the navigation bar is an illustration of a person in a white shirt standing next to a list of input fields. The main form area includes a 'username' label above a text input field, a 'Password' label above a text input field, and a 'Confirm Password' label above a text input field. A 'Signup' button is positioned at the bottom of the form.

Login Sign Up

username

Password

Confirm Password

Signup

Search

Pagina complessa che permette di cercare eventi sportivi con filtri e due modalità di visualizzazione (mappa e lista).

```
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!);
    });
  }

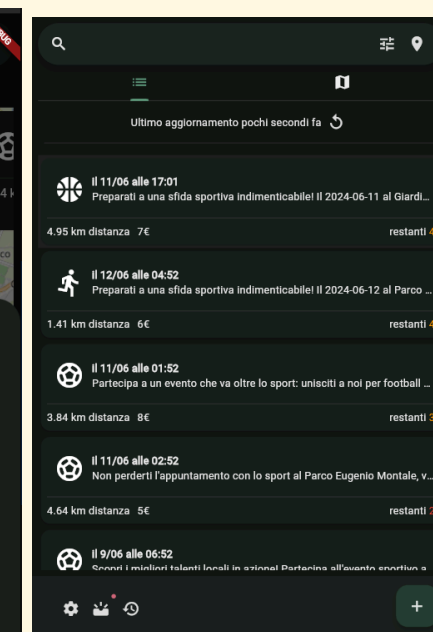
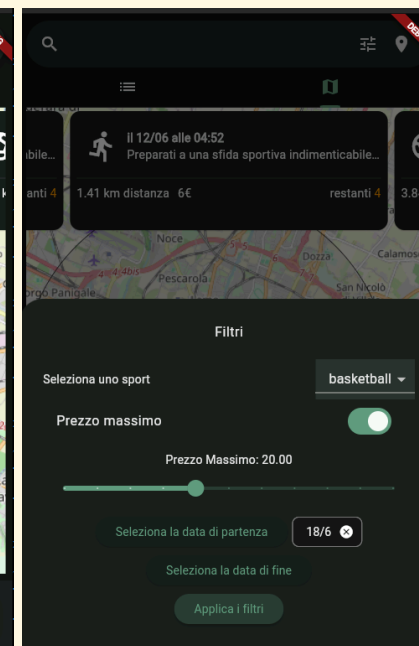
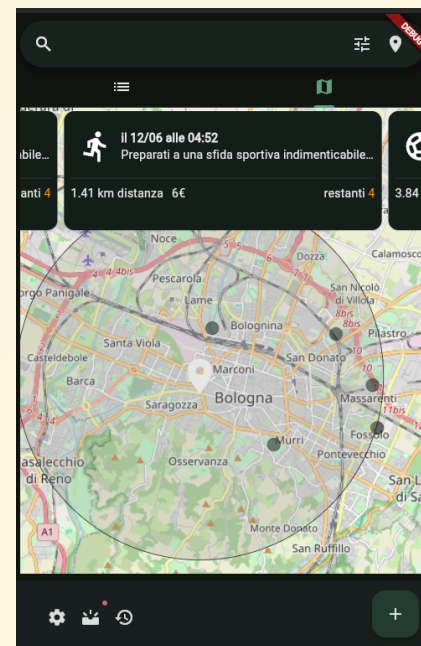
  @override
  Widget build(BuildContext context) {
    return Consumer<DataProvider>({builder: (context, dataProvider, child) {
      return SearchPage(key: UniqueKey(), data: dataProvider.toApplicationData());
    }});
  }
}
```

```
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();
}
```

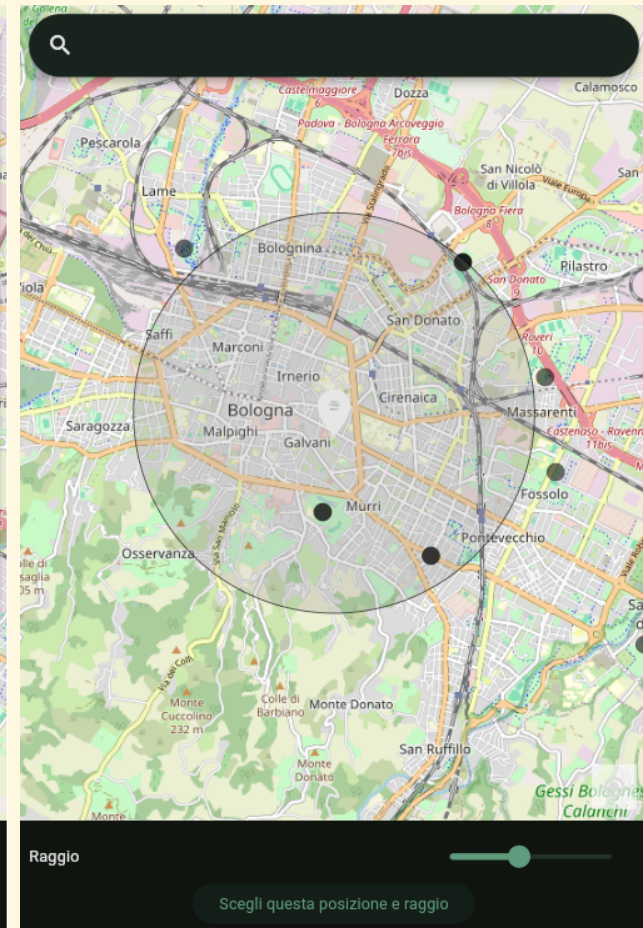
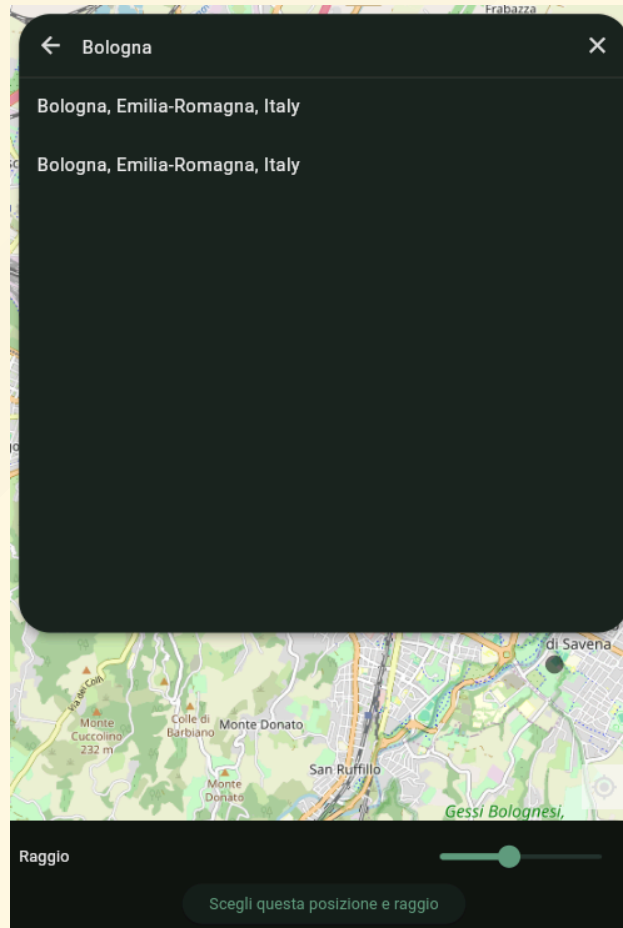
Modalità di Visualizzazione

- **Mappa:** Visualizzazione eventi tramite marker
- **Lista:** Visualizzazione dettagliata degli eventi



Cambia raggio

Questa activity serve per cambiare il raggio di ricerca della search.



Aggiungi attività

Compilazione di diverse pagine per creare un'attività sportiva:

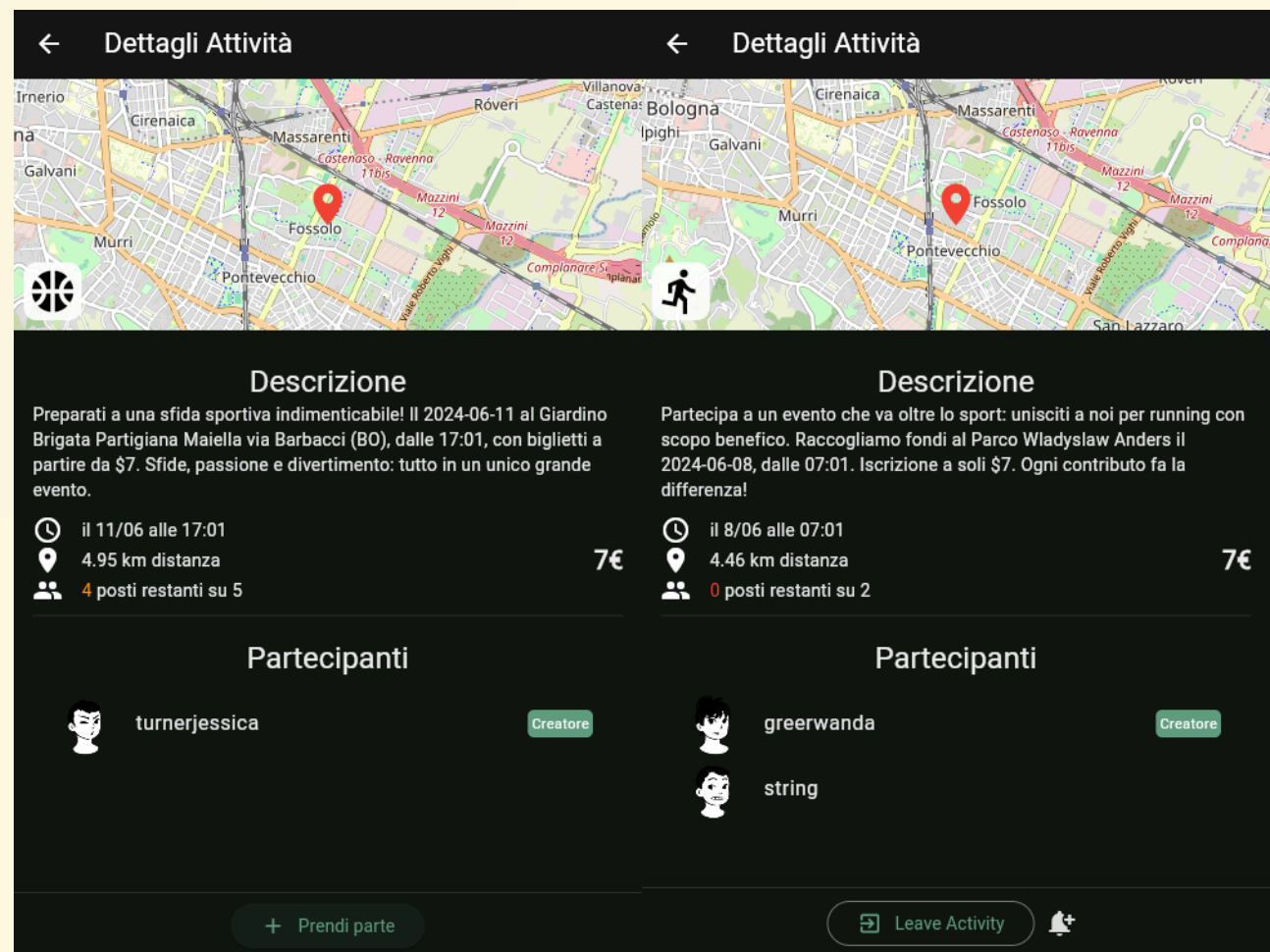
- **Posizione:** Selezione posizione evento
- **Data:** Selezione data e ora evento
- **Dettagli:** Descrizione evento, disciplina sportiva, partecipanti, prezzo

The image displays three sequential mobile app screens for adding a sports activity, each with a dark background and a back arrow in the top-left corner.

- Screen 1: Scegli un luogo**
Title: Scegli un luogo
Subtitle: Scegli il luogo dell'attività
Text: Scegli una posizione
Bottom navigation: Left arrow, Right arrow
- Screen 2: Scegli la data**
Title: Scegli la data
Subtitle: Scegli la data e l'ora dell'attività
Text: Select Date
Text: Data selezionata: il 7/06 alle 16:40
Bottom navigation: Left arrow, Right arrow
- Screen 3: Aggiungi i dettagli**
Title: Aggiungi i dettagli
Subtitle: Aggiungi i dettagli dell'attività
Form fields:
 - Descrizione: basket con gli amichetti
 - Sport: basketball (dropdown menu)
 - È gratis?: ☒ (toggle switch)
 - Numero di persone: (text input)Bottom navigation: Left arrow, Right arrow

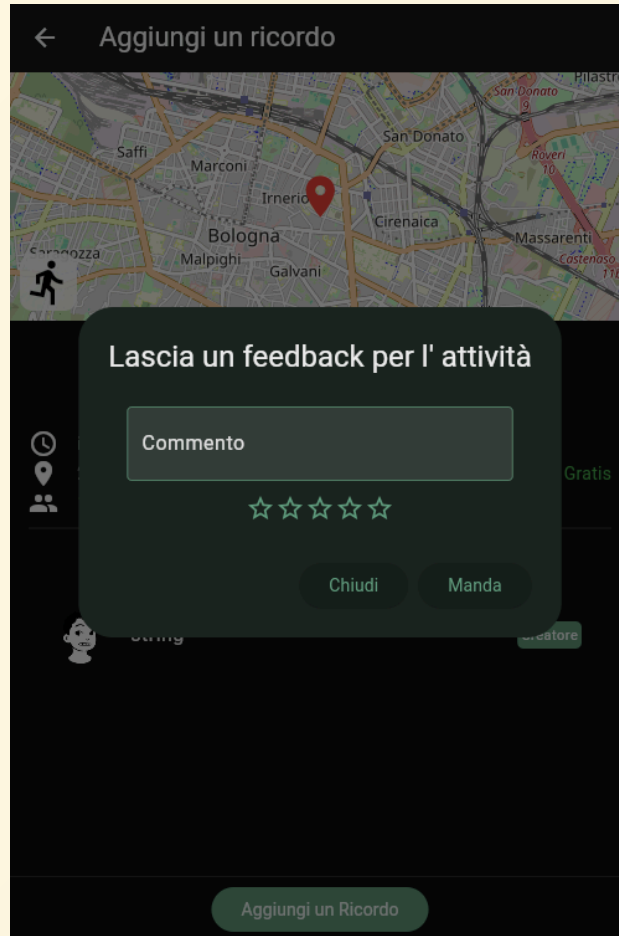
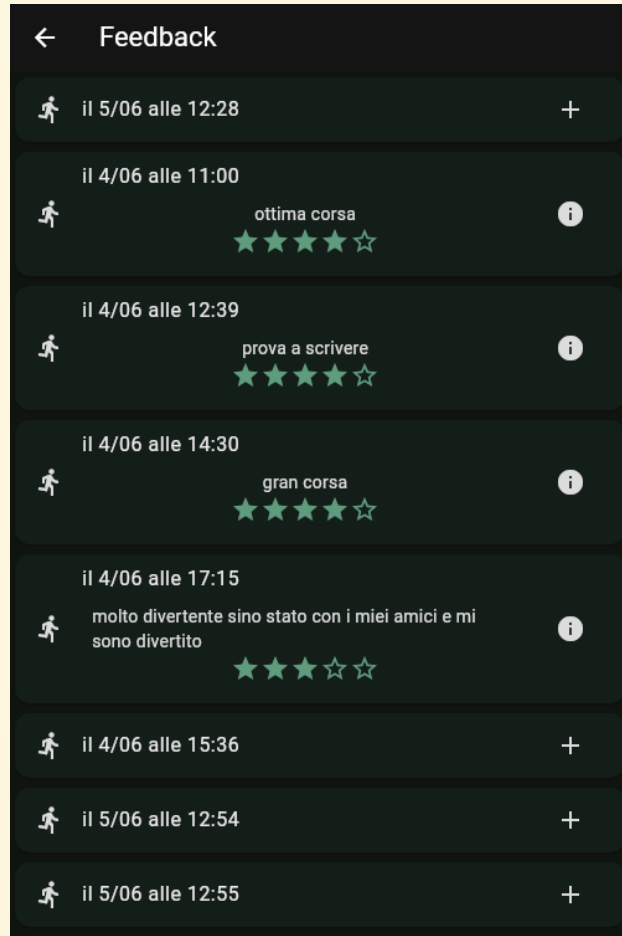
Partecipa ad un'attività

Cliccare sul bottone "partecipa" per aggiungersi alla lista dei partecipanti e impostare un promemoria per l'evento, tramite il bottone di notifica.



Storia

Visualizzare eventi passati e aggiungere feedback.



Impostazioni

Modifica delle informazioni utente e delle notifiche evento. Disconnessione dall'account e eliminazione dati locali.

