import 'package:flutter/material.dart';

import 'package:shared\_preferences/shared\_preferences.dart';

import 'dart:convert';

import 'package:provider/provider.dart';

import 'cigarette\_counter.dart';

import 'modify.dart';

import 'plots.dart';

import 'homePage.dart';

class ProfilePage extends StatefulWidget {

  final String accountName;

  ProfilePage({required this.accountName});

  @override

  \_ProfilePageState createState() => \_ProfilePageState();

}

class \_ProfilePageState extends State<ProfilePage> {

  String? \_cigaretteType;

  double? \_nicotine;

  String? \_registrationDate;

  @override

  void initState() {

    super.initState();

    \_loadUserData();

    \_loadCigarettesSmokedToday();

  }

  Future<void> \_loadUserData() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String? accountName = prefs.getString('loggedInAccount');

    if (accountName != null) {

      String? usersData = prefs.getString('users');

      Map<String, dynamic> users = usersData != null ? json.decode(usersData) : {};

      if (users.containsKey(accountName)) {

        setState(() {

          \_cigaretteType = users[accountName]['CigaretteType'];

          \_nicotine = users[accountName]['Nicotine'];

          \_registrationDate = users[accountName]['registrationDate'];

        });

      }

    }

  }

  Future<void> \_loadCigarettesSmokedToday() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String todayKey = \_getTodayKey();

    String dailyCountsKey = "${widget.accountName}\_dailyCounts";

    String? dailyCountsData = prefs.getString(dailyCountsKey);

    Map<String, int> dailyCounts = dailyCountsData != null ? Map<String, int>.from(json.decode(dailyCountsData)) : {};

    int cigarettes = dailyCounts[todayKey] ?? 0;

    Provider.of<CigaretteCounter>(context, listen: false).setCigarettes(cigarettes);

  }

  String \_getTodayKey() {

    DateTime now = DateTime.now();

    String accountName = widget.accountName;

    return "${accountName}\_cigarettes\_${now.year}${now.month}${now.day}";

  }

  Future<void> \_incrementCigaretteCount() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String todayKey = \_getTodayKey();

    int newCount = Provider.of<CigaretteCounter>(context, listen: false).cigarettesSmokedToday + 1;

    setState(() {

      prefs.setInt(todayKey, newCount);

      Provider.of<CigaretteCounter>(context, listen: false).incrementCigarettes();

    });

    \_saveDailyCount(newCount);  // Salva il conteggio giornaliero

    \_loadUserData();

  }

  Future<void> \_decrementCigaretteCount() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String todayKey = \_getTodayKey();

    int currentCount = Provider.of<CigaretteCounter>(context, listen: false).cigarettesSmokedToday;

    if (currentCount > 0) {

      int newCount = currentCount - 1;

      setState(() {

        prefs.setInt(todayKey, newCount);

        Provider.of<CigaretteCounter>(context, listen: false).setCigarettes(newCount);

      });

      \_saveDailyCount(newCount);  // Salva il conteggio giornaliero

    }

  }

  Future<void> \_saveDailyCount(int count) async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String todayKey = \_getTodayKey();

    String dailyCountsKey = "${widget.accountName}\_dailyCounts";

    String? dailyCountsData = prefs.getString(dailyCountsKey);

    Map<String, int> dailyCounts = dailyCountsData != null ? Map<String, int>.from(json.decode(dailyCountsData)) : {};

    dailyCounts[todayKey] = count;

    await prefs.setString(dailyCountsKey, json.encode(dailyCounts));

  }

  Future<void> \_logout() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    await prefs.remove('loggedInAccount'); // Remove the login state

    Navigator.pushAndRemoveUntil(

      context,

      MaterialPageRoute(builder: (context) => HomePage()), // Replace Homepage() with your actual homepage widget

      (Route<dynamic> route) => false, // Remove all previous routes

    );

  }

  @override

  Widget build(BuildContext context) {

    final cigaretteProvider = Provider.of<CigaretteCounter>(context);

    return Scaffold(

      appBar: AppBar(title: Text('Profile')),

      body: Center(

        child: Column(

          mainAxisAlignment: MainAxisAlignment.center,

          children: <Widget>[

            Text('Account Name: ${widget.accountName}'),

            Text('Cigarette Type: $\_cigaretteType'),

            Text('Nicotine: $\_nicotine'),

            SizedBox(height: 20),

            Text('Registration Date: ${\_registrationDate != null ? DateTime.parse(\_registrationDate!).toLocal().toString() : 'Not Available'}'),

            SizedBox(height: 20),

            ElevatedButton(

              onPressed: () {

                Navigator.push(

                  context,

                  MaterialPageRoute(

                    builder: (context) => ModifyPage(

                      accountName: widget.accountName,

                      cigaretteType: \_cigaretteType!,

                      nicotine: \_nicotine!,

                    ),

                  ),

                ).then((\_) {

                  \_loadUserData();

                });

              },

              child: Text('Modify Profile'),

            ),

            SizedBox(height: 20),

            Row(

              mainAxisAlignment: MainAxisAlignment.center,

              children: <Widget>[

                GestureDetector(

                  onTap: () {

                    \_decrementCigaretteCount();

                  },

                  child: Container(

                    padding: EdgeInsets.all(0),

                    decoration: BoxDecoration(

                      shape: BoxShape.circle,

                      border: Border.all(color: Colors.grey, width: 2),

                    ),

                    child: Icon(

                      Icons.remove,

                      color: Colors.grey,

                    ),

                  ),

                ),

                SizedBox(width: 10),

                ElevatedButton(

                  onPressed: () {

                    \_incrementCigaretteCount();

                  },

                  child: Text('Add a Cigarette'),

                ),

                SizedBox(width: 20),

                Text(

                  '${cigaretteProvider.cigarettesSmokedToday}',

                  style: TextStyle(fontSize: 24),

                ),

              ],

            ),

            SizedBox(height: 20),

            ElevatedButton(

              onPressed: () {

                Navigator.push(

                  context,

                  MaterialPageRoute(

                    builder: (context) => Plots(accountName: widget.accountName),

                  ),

                );

              },

              child: Text('View Progress'),

            ),

          ],

        ),

      ),

      floatingActionButton: FloatingActionButton(

        onPressed: \_logout,

        child: Icon(Icons.logout),

        tooltip: 'Logout',

      ),

    );

  }

}

profilePage.dart (VERSIONE CON I METODI IN PIU’)

import 'package:flutter/material.dart';

import 'package:shared\_preferences/shared\_preferences.dart';

import 'dart:convert';

import 'package:provider/provider.dart';

import 'cigarette\_counter.dart';

import 'modify.dart';

import 'plots.dart';

import 'homePage.dart';

import 'delete\_account\_page.dart';

class ProfilePage extends StatefulWidget {

  final String accountName;

  ProfilePage({required this.accountName});

  @override

  \_ProfilePageState createState() => \_ProfilePageState();

}

class \_ProfilePageState extends State<ProfilePage> {

  String? \_cigaretteType;

  double? \_nicotine;

  String? \_registrationDate;

  @override

  void initState() {

    super.initState();

    \_loadUserData();

    \_loadCigarettesSmokedToday();

    \_checkSavedData();

  }

  Future<void> \_loadUserData() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String? accountName = prefs.getString('loggedInAccount');

    if (accountName != null) {

      String? usersData = prefs.getString('users');

      Map<String, dynamic> users = usersData != null ? json.decode(usersData) : {};

      if (users.containsKey(accountName)) {

        setState(() {

          \_cigaretteType = users[accountName]['CigaretteType'];

          \_nicotine = users[accountName]['Nicotine'];

          \_registrationDate = users[accountName]['registrationDate'];

        });

      }

    }

  }

  Future<void> \_loadCigarettesSmokedToday() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String todayKey = \_getTodayKey();

    int cigarettes = prefs.getInt(todayKey) ?? 0;

    Provider.of<CigaretteCounter>(context, listen: false).setCigarettes(cigarettes);

  }

  String \_getTodayKey() {

    DateTime now = DateTime.now();

    String accountName = widget.accountName;

    return "${accountName}\_cigarettes\_${now.year}${now.month}${now.day}";

  }

Future<void> \_incrementCigaretteCount() async {

  SharedPreferences prefs = await SharedPreferences.getInstance();

  // Recupera i dati dell'utente

  String accountName = widget.accountName;

  String todayKey = \_getTodayKey();

  int newCount = Provider.of<CigaretteCounter>(context, listen: false).cigarettesSmokedToday + 1;

  setState(() {

    // Salva il conteggio delle sigarette per oggi

    prefs.setInt(todayKey, newCount);

    Provider.of<CigaretteCounter>(context, listen: false).incrementCigarettes();

  });

  // Registra l'orario della sigaretta e aggiorna i dati

  await \_recordCigaretteTime();

  // Aggiungi il conteggio giornaliero

  await \_updateDailyCounts();

}

Future<void> \_updateDailyCounts() async {

  SharedPreferences prefs = await SharedPreferences.getInstance();

  String accountName = widget.accountName;

  String dailyCountsKey = "${accountName}\_dailyCounts";

  String? dailyCountsData = prefs.getString(dailyCountsKey);

  // Carica i dati esistenti o inizializza una nuova mappa

  Map<String, int> dailyCounts = dailyCountsData != null

    ? Map<String, int>.from(json.decode(dailyCountsData))

    : {};

  // Ottieni la chiave per il giorno corrente e il conteggio di sigarette

  String todayKey = \_getTodayKey();

  int todayCount = Provider.of<CigaretteCounter>(context, listen: false).cigarettesSmokedToday;

  // Aggiorna il conteggio per il giorno corrente

  dailyCounts[todayKey] = todayCount;

  // Salva i conteggi giornalieri

  prefs.setString(dailyCountsKey, json.encode(dailyCounts));

}

  Future<void> \_checkSavedData() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String dailyCountsKey = "${widget.accountName}\_dailyCounts";

    String? dailyCountsData = prefs.getString(dailyCountsKey);

    print("Saved dailyCountsData: $dailyCountsData");

  }

  Future<void> \_recordCigaretteTime() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String currentKey = "${widget.accountName}\_${DateTime.now().toIso8601String()}";

    prefs.setInt(currentKey, 1); // Registra che è stata fumata una sigaretta a questo orario

  }

  Future<void> \_decrementCigaretteCount() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String todayKey = \_getTodayKey();

    int currentCount = Provider.of<CigaretteCounter>(context, listen: false).cigarettesSmokedToday;

    if (currentCount > 0) {

      int newCount = currentCount - 1;

      setState(() {

        prefs.setInt(todayKey, newCount);

        Provider.of<CigaretteCounter>(context, listen: false).setCigarettes(newCount);

      });

    }

    // Aggiungi il conteggio giornaliero

    await \_updateDailyCounts();

  }

  Future<void> \_logout() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    await prefs.remove('loggedInAccount'); // Remove the login state

    Navigator.pushAndRemoveUntil(

      context,

      MaterialPageRoute(builder: (context) => HomePage()), // Replace HomePage() with your actual homepage widget

      (Route<dynamic> route) => false, // Remove all previous routes

    );

  }

  void \_showDeleteConfirmation() {

    Navigator.push(

      context,

      MaterialPageRoute(

        builder: (context) => DeleteAccountPage(

          onDelete: () async {

            SharedPreferences prefs = await SharedPreferences.getInstance();

            String? accountName = prefs.getString('loggedInAccount');

            if (accountName != null) {

              // Recupera i dati degli utenti

              String? usersData = prefs.getString('users');

              Map<String, dynamic> users = usersData != null ? json.decode(usersData) : {};

              // Rimuovi solo i dati dell'utente specifico

              if (users.containsKey(accountName)) {

                users.remove(accountName); // Rimuovi solo i dati dell'utente corrente

                await prefs.setString('users', json.encode(users));

              }

              // Rimuovi il login

              await prefs.remove('loggedInAccount');

            }

            Navigator.pushAndRemoveUntil(

              context,

              MaterialPageRoute(builder: (context) => HomePage()),

              (Route<dynamic> route) => false,

            );

          },

          onCancel: () {

            Navigator.pop(context); // Torna alla pagina ProfilePage

          },

        ),

      ),

    );

  } //\_showDeleteConfirmation

  @override

  void dispose() {

    // Salva i conteggi quando il widget viene distrutto

    \_updateDailyCounts();

    super.dispose();

  }

  @override

  Widget build(BuildContext context) {

    final cigaretteProvider = Provider.of<CigaretteCounter>(context);

    return Scaffold(

      appBar: AppBar(title: Text('Profile')),

      body: Center(

        child: Column(

          mainAxisAlignment: MainAxisAlignment.center,

          children: <Widget>[

            Text('Account Name: ${widget.accountName}'),

            Text('Cigarette Type: $\_cigaretteType'),

            Text('Nicotine: $\_nicotine'),

            SizedBox(height: 20),

            Text('Registration Date: ${\_registrationDate != null ? DateTime.parse(\_registrationDate!).toLocal().toString() : 'Not Available'}'),

            SizedBox(height: 20),

            ElevatedButton(

              onPressed: () {

                Navigator.push(

                  context,

                  MaterialPageRoute(

                    builder: (context) => ModifyPage(

                      accountName: widget.accountName,

                      cigaretteType: \_cigaretteType!,

                      nicotine: \_nicotine!,

                    ),

                  ),

                ).then((\_) {

                  \_loadUserData();

                });

              },

              child: Text('Modify Profile'),

            ),

            SizedBox(height: 20),

            Row(

              mainAxisAlignment: MainAxisAlignment.center,

              children: <Widget>[

                GestureDetector(

                  onTap: () {

                    \_decrementCigaretteCount();

                  },

                  child: Container(

                    padding: EdgeInsets.all(0),

                    decoration: BoxDecoration(

                      shape: BoxShape.circle,

                      border: Border.all(color: Colors.grey, width: 2),

                    ),

                    child: Icon(

                      Icons.remove,

                      color: Colors.grey,

                    ),

                  ),

                ),

                SizedBox(width: 10),

                ElevatedButton(

                  onPressed: () {

                    \_incrementCigaretteCount();

                  },

                  child: Text('Add a Cigarette'),

                ),

                SizedBox(width: 20),

                Text(

                  '${cigaretteProvider.cigarettesSmokedToday}',

                  style: TextStyle(fontSize: 24),

                ),

              ],

            ),

            SizedBox(height: 20),

            ElevatedButton(

              onPressed: () {

                Navigator.push(

                  context,

                  MaterialPageRoute(

                    builder: (context) => Plots(accountName: widget.accountName),

                  ),

                );

              },

              child: Text('View Progress'),

            ),

          ],

        ),

      ),

      bottomNavigationBar: Padding(

        padding: EdgeInsets.all(16),

        child: Row(

          mainAxisAlignment: MainAxisAlignment.spaceBetween,

          children: <Widget>[

            ElevatedButton(

              onPressed: \_showDeleteConfirmation,

              child: Icon(Icons.delete),

              style: ElevatedButton.styleFrom(

                shadowColor: Colors.red,

                padding: EdgeInsets.symmetric(horizontal: 20),

              ),

            ),

            ElevatedButton(

              onPressed: \_logout,

              child: Icon(Icons.logout),

              style: ElevatedButton.styleFrom(

                shadowColor: Colors.blue,

                padding: EdgeInsets.symmetric(horizontal: 20),

              ),

            ),

          ],

        ),

      ),

    );

  }

}

Plots.dart:

import 'dart:math';

import 'package:flutter/material.dart';

import 'package:shared\_preferences/shared\_preferences.dart';

import 'dart:convert';

import 'package:provider/provider.dart';

import 'cigarette\_counter.dart';

import 'package:progetto/charts/plot\_creation.dart';

class Plots extends StatefulWidget {

  final String accountName;

  Plots({required this.accountName});

  @override

  \_PlotsState createState() => \_PlotsState();

}

class \_PlotsState extends State<Plots> {

  DateTime? registrationDate;

  List<NicotineLevel> data = [];

  List<HourlyNicotineLevel> hourlyData = [];

  int \_cigarettesPerDay = 0;

  int threshold = 0;

  bool isLoading = true;

  @override

  void initState() {

    super.initState();

    \_loadRegistrationData();

  }

  Future<void> \_loadRegistrationData() async {

    try {

      SharedPreferences prefs = await SharedPreferences.getInstance();

      String? usersData = prefs.getString('users');

      Map<String, dynamic> users = usersData != null ? json.decode(usersData) : {};

      if (users.containsKey(widget.accountName)) {

        var userProfile = users[widget.accountName];

        String? dateStr = userProfile['registrationDate'];

        \_cigarettesPerDay = userProfile['CigarettesPerDay'] ?? 0;

        threshold = \_cigarettesPerDay; //threshold initialization

        if (dateStr != null) {

          registrationDate = DateTime.parse(dateStr);

        } else {

          registrationDate = DateTime.now();

        }

        // Calcola quanti giorni sono passati dalla registrazione

        if (registrationDate != null) {

          int daysSinceRegistration = DateTime.now().difference(registrationDate!).inDays;

          // Decrementa la soglia di 1 per ogni 7 giorni passati

          threshold -= (daysSinceRegistration ~/ 7);

          if (threshold < 0) threshold = 0; // La soglia non può andare sotto 0

        }

        await \_generateChartData(users);

        await \_generateHourlyData();

      } else {

        print("User not found: ${widget.accountName}");

      }

    } catch (e) {

      print("Error loading registration data: $e");

    } finally {

      setState(() {

        isLoading = false;

      });

    }

  }

  Future<void> \_generateChartData(Map<String, dynamic> users) async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String dailyCountsKey = "${widget.accountName}\_dailyCounts";

    String? dailyCountsData = prefs.getString(dailyCountsKey);

    Map<String, int> dailyCounts = dailyCountsData != null ? Map<String, int>.from(json.decode(dailyCountsData)) : {};

    if (registrationDate != null) {

      DateTime startDate = registrationDate!;

      data = [];

      DateTime now = DateTime.now();

      int totalCigarettes = 0;

      //int daysToGenerate = \_cigarettesPerDay\*7;

      int daysToGenerate = now.difference(startDate).inDays + 1;

      for (int i = 0; i < daysToGenerate; i++) {

        DateTime currentDate = startDate.add(Duration(days: i));

        String dateKey = "${widget.accountName}\_cigarettes\_${currentDate.year}${currentDate.month}${currentDate.day}";

        double cigarettes = dailyCounts[dateKey]?.toDouble() ?? 0.0;

        totalCigarettes += cigarettes.toInt();

        data.add(NicotineLevel(date: currentDate, level: cigarettes));

      }

      final cigaretteCounter = Provider.of<CigaretteCounter>(context, listen: false);

      data.add(NicotineLevel(date: now, level: cigaretteCounter.cigarettesSmokedToday.toDouble()));

      totalCigarettes += cigaretteCounter.cigarettesSmokedToday;

      int futureDays = totalCigarettes \* 7;

      for (int i = 1; i <= futureDays; i++) {

        DateTime futureDate = now.add(Duration(days: i));

        data.add(NicotineLevel(date: futureDate, level: 0.0));

      }

      setState(() {});

    }

  }

  Future<void> \_generateHourlyData() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    DateTime now = DateTime.now();

    DateTime startOfDay = DateTime(now.year, now.month, now.day);

    hourlyData = [];

    // Mantieni un registro cumulativo delle curve

    Map<DateTime, double> cumulativeNicotineLevels = {};

    // Ottieni il tipo di sigaretta e il livello di nicotina salvati per l'utente corrente

    String? usersData = prefs.getString('users');

    Map<String, dynamic> users = usersData != null ? json.decode(usersData) : {};

    double nicotinePerCigarette = 1.0; // Valore predefinito nel caso non ci siano dati

    if (users.containsKey(widget.accountName)) {

      var userProfile = users[widget.accountName];

      nicotinePerCigarette = userProfile['Nicotine'] ?? 1.0;

    }

    Set<String> keys = prefs.getKeys();

    // Raccogli tutte le sigarette fumate dal salvataggio

    for (String key in keys) {

      if (key.startsWith("${widget.accountName}\_cigarette\_")) {

        DateTime timestamp = DateTime.parse(key.split('\_')[2]);

        DateTime hour = DateTime(timestamp.year, timestamp.month, timestamp.day, timestamp.hour);

        // Aggiungi la curva di nicotina per questa sigaretta al registro cumulativo

        for (int i = 0; i < 24; i++) {

          DateTime currentHour = hour.add(Duration(hours: i));

          double nicotineLevel = nicotinePerCigarette / pow(2, i \* 60 / 90); // Effetto nel tempo

          if (cumulativeNicotineLevels.containsKey(currentHour)) {

            cumulativeNicotineLevels[currentHour] = cumulativeNicotineLevels[currentHour]! + nicotineLevel;

          } else {

            cumulativeNicotineLevels[currentHour] = nicotineLevel;

          }

        }

      }

    }

    // Aggiungi l'ultima sigaretta appena fumata

    final cigaretteCounter = Provider.of<CigaretteCounter>(context, listen: false);

    if (cigaretteCounter.cigarettesSmokedToday > 0) {

      DateTime lastCigaretteTime = now;

      DateTime hour = DateTime(lastCigaretteTime.year, lastCigaretteTime.month, lastCigaretteTime.day, lastCigaretteTime.hour);

      // Aggiorna solo le ore successive alla nuova sigaretta

      for (int i = 0; i < 24; i++) {

        DateTime currentHour = hour.add(Duration(hours: i));

        double nicotineLevel = nicotinePerCigarette / pow(2, i \* 60 / 90);

        nicotineLevel \*= cigaretteCounter.cigarettesSmokedToday;

        if (cumulativeNicotineLevels.containsKey(currentHour)) {

          cumulativeNicotineLevels[currentHour] = cumulativeNicotineLevels[currentHour]! + nicotineLevel;

        } else {

          cumulativeNicotineLevels[currentHour] = nicotineLevel;

        }

      }

    }

    // Ora converte cumulativeNicotineLevels in hourlyData

    for (int i = 0; i < 24; i++) {

      DateTime hour = startOfDay.add(Duration(hours: i));

      double nicotineLevel = cumulativeNicotineLevels[hour] ?? 0.0;

      hourlyData.add(HourlyNicotineLevel(time: hour, level: nicotineLevel));

    }

    setState(() {});

    //PROVA PER VALIDAZIONE- IN CASO TOGLIERE

    for (var hourly in hourlyData) {

      print('Ora: ${hourly.time}, Livello di nicotina: ${hourly.level}');

    }

  }

  @override

  Widget build(BuildContext context) {

    final cigaretteCounter = Provider.of<CigaretteCounter>(context);

    DateTime now = DateTime.now();

    return Scaffold(

      appBar: AppBar(title: Text('Plots')),

      body: isLoading

          ? Center(child: CircularProgressIndicator())

          : Padding(

              padding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 8.0),

              child: Column(

                children: <Widget>[

                  Padding(

                    padding: const EdgeInsets.all(16.0),

                    child: Row(

                      mainAxisAlignment: MainAxisAlignment.start,

                      children: [

                        Text(

                          'Cigarettes smoked today: ${cigaretteCounter.cigarettesSmokedToday}/$threshold',

                          style: TextStyle(fontSize: 20),

                        ),

                      ],

                    ),

                  ),

                  Container(

                    height: MediaQuery.of(context).size.height \* 0.3,

                    child: SingleChildScrollView(

                      scrollDirection: Axis.horizontal,

                      child: Container(

                        width: (\_cigarettesPerDay \* 7 \* 50.0),

                        child: NicotineChart(

                          NicotineChart.createSampleData(data),

                          animate: true,

                          registrationDate: registrationDate!,

                          cigarettesPerDay: \_cigarettesPerDay,

                        ),

                      ),

                    ),

                  ),

                  Padding(

                    padding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 8.0),

                    child: Align(

                      alignment: Alignment.centerLeft,

                      child: Text(

                        'Slide horizontally to view more data',

                        style: TextStyle(fontSize: 16),

                      ),

                    ),

                  ),

                  SizedBox(height: 20), // Spazio tra i grafici

                  Padding(

                    padding: const EdgeInsets.all(8.0),

                    child: Text(

                      '${now.day.toString().padLeft(2, '0')}/${now.month.toString().padLeft(2, '0')}/${now.year}',

                      style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),

                    ),

                  ),

                  Container(

                    height: MediaQuery.of(context).size.height \* 0.3, // Altezza ridotta

                    child: HourlyNicotineChart(

                      HourlyNicotineChart.createSampleData(hourlyData),

                      animate: true,

                    ),

                  ),

                ],

              ),

            ),

    );

  }

}

GENERATE CHART DATA in plots.dart

  Future<void> \_generateChartData(Map<String, dynamic> users) async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String dailyCountsKey = "${widget.accountName}\_dailyCounts";

    String? dailyCountsData = prefs.getString(dailyCountsKey);

    Map<String, int> dailyCounts = dailyCountsData != null ? Map<String, int>.from(json.decode(dailyCountsData)) : {};

    if (registrationDate != null) {

      DateTime startDate = registrationDate!;

      data = [];

      DateTime now = DateTime.now();

      int totalCigarettes = 0;

      //int daysToGenerate = \_cigarettesPerDay\*7;

      int daysToGenerate = now.difference(startDate).inDays + 1;

      for (int i = 0; i < daysToGenerate; i++) {

        DateTime currentDate = startDate.add(Duration(days: i));

        String dateKey = "${widget.accountName}\_cigarettes\_${currentDate.year}${currentDate.month}${currentDate.day}";

        double cigarettes = dailyCounts[dateKey]?.toDouble() ?? 0.0;

        totalCigarettes += cigarettes.toInt();

        data.add(NicotineLevel(date: currentDate, level: cigarettes));

      }

      final cigaretteCounter = Provider.of<CigaretteCounter>(context, listen: false);

      data.add(NicotineLevel(date: now, level: cigaretteCounter.cigarettesSmokedToday.toDouble()));

      totalCigarettes += cigaretteCounter.cigarettesSmokedToday;

      int futureDays = totalCigarettes \* 7;

      for (int i = 1; i <= futureDays; i++) {

        DateTime futureDate = now.add(Duration(days: i));

        data.add(NicotineLevel(date: futureDate, level: 0.0));

      }

      setState(() {});

    }

  }

\_LOAD CIGARETTES SMOKED TODAY in profilePage.dart:

 Future<void> \_loadCigarettesSmokedToday() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String todayKey = \_getTodayKey();

    String dailyCountsKey = "${widget.accountName}\_dailyCounts";

    String? dailyCountsData = prefs.getString(dailyCountsKey);

    //Map<String, int> dailyCounts = dailyCountsData != null ? Map<String, int>.from(json.decode(dailyCountsData)) : {};

    int cigarettes = prefs.getInt(todayKey) ?? 0;

    Provider.of<CigaretteCounter>(context, listen: false).setCigarettes(cigarettes);

  }

INCREMENT CIGARETTE COUNT in profilePage.dart:

  Future<void> \_incrementCigaretteCount() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    // Recupera i dati dell'utente

    //String accountName = widget.accountName;

    String todayKey = \_getTodayKey();

    String hourlyKey = \_getHourlyKey();

    String hourlyNicotineKey = \_getHourlyNicotineKey();

    int newCount = Provider.of<CigaretteCounter>(context, listen: false).cigarettesSmokedToday + 1;

    setState(() {

      // Salva il conteggio delle sigarette per oggi

      prefs.setInt(todayKey, newCount);

      Provider.of<CigaretteCounter>(context, listen: false).incrementCigarettes();

    });

    // Registra l'orario della sigaretta e aggiorna i dati

    await \_recordCigaretteTime();

    \_saveDailyCount(newCount);  // Salva il conteggio giornaliero

    // Incrementa il contatore orario

    int hourlyCount = prefs.getInt(hourlyKey) ?? 0;

    hourlyCount++;

    double hourlyNicotine = prefs.getDouble(hourlyNicotineKey) ?? 0.0;

    hourlyNicotine += \_nicotine ?? 0.0;

    Provider.of<CigaretteCounter>(context, listen: false).updateHourlyCount(hourlyCount, hourlyNicotine);

    prefs.setInt(hourlyKey, hourlyCount);

    prefs.setDouble(hourlyNicotineKey, hourlyNicotine);

    // Aggiorna la UI per riflettere il nuovo valore del contatore orario

    setState(() {});

  }

GENERATE HOURLY DATA in plots.dart:

Future<void> \_generateHourlyData() async {

SharedPreferences prefs = await SharedPreferences.getInstance();

DateTime now = DateTime.now();

DateTime startOfDay = DateTime(now.year, now.month, now.day);

hourlyData = [];

final cigaretteCounter = Provider.of<CigaretteCounter>(context, listen: false);

int hourlyCigarettesSmoked = cigaretteCounter.hourlyCigarettesSmoked;

double hourlyNicotine = cigaretteCounter.hourlyNicotine;

// Usa una mappa per memorizzare il livello massimo di nicotina per ogni ora

Map<DateTime, double> hourlyNicotineLevels = {};

Set<String> keys = prefs.getKeys();

for (String key in keys) {

if (key.startsWith("${widget.accountName}cigarette")) {

DateTime timestamp = DateTime.parse(key.split('\_')[2]);

DateTime hour = DateTime(timestamp.year, timestamp.month, timestamp.day, timestamp.hour);

double nicotineLevel = hourlyNicotine;

if (hourlyNicotineLevels.containsKey(hour)) {

// Aggiorna solo se il nuovo livello di nicotina è più alto

hourlyNicotineLevels[hour] = hourlyNicotineLevels[hour]! < nicotineLevel ? nicotineLevel : hourlyNicotineLevels[hour]!;

} else {

hourlyNicotineLevels[hour] = nicotineLevel;

}

}

}

if (cigaretteCounter.hourlyCigarettesSmoked > 0) {

DateTime lastCigaretteTime = now;

DateTime hour = DateTime(lastCigaretteTime.year, lastCigaretteTime.month, lastCigaretteTime.day, lastCigaretteTime.hour);

double nicotineLevel = hourlyNicotine;

if (hourlyNicotineLevels.containsKey(hour)) {

hourlyNicotineLevels[hour] = hourlyNicotineLevels[hour]! < nicotineLevel ? nicotineLevel : hourlyNicotineLevels[hour]!;

} else {

hourlyNicotineLevels[hour] = nicotineLevel;

}

}

for (int i = 0; i < 24; i++) {

DateTime hour = startOfDay.add(Duration(hours: i));

double nicotineLevel = hourlyNicotineLevels[hour] ?? 0.0;

hourlyData.add(HourlyNicotineLevel(time: hour, level: nicotineLevel));

}

print("Hourly Data: $hourlyData"); // Debug print

setState(() {});

}

\_generateChartData:

  Future<void> \_generateChartData(Map<String, dynamic> users) async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String dailyCountsKey = "${widget.accountName}\_dailyCounts";

    String? dailyCountsData = prefs.getString(dailyCountsKey);

    Map<String, int> dailyCounts = dailyCountsData != null ? Map<String, int>.from(json.decode(dailyCountsData)) : {};

    if (registrationDate != null) {

      DateTime startDate = registrationDate!;

      data = [];

      DateTime now = DateTime.now();

      int totalCigarettes = 0;

      //int daysToGenerate = \_cigarettesPerDay\*7;

      int daysToGenerate = now.difference(startDate).inDays + 1;

      for (int i = 0; i < daysToGenerate; i++) {

        DateTime currentDate = startDate.add(Duration(days: i));

        String dateKey = "${widget.accountName}\_cigarettes\_${currentDate.year}${currentDate.month}${currentDate.day}";

        double cigarettes = dailyCounts[dateKey]?.toDouble() ?? 0.0;

        totalCigarettes += cigarettes.toInt();

        data.add(NicotineLevel(date: currentDate, level: cigarettes));

      }

      final cigaretteCounter = Provider.of<CigaretteCounter>(context, listen: false);

      data.add(NicotineLevel(date: now, level: cigaretteCounter.cigarettesSmokedToday.toDouble()));

      totalCigarettes += cigaretteCounter.cigarettesSmokedToday;

      int futureDays = totalCigarettes \* 7;

      for (int i = 1; i <= futureDays; i++) {

        DateTime futureDate = now.add(Duration(days: i));

        data.add(NicotineLevel(date: futureDate, level: 0.0));

      }

      setState(() {});

    }

  }

GENERATE CHART DATA

  Future<void> \_generateChartData(Map<String, dynamic> users) async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String dailyCountsKey = "${widget.accountName}\_dailyCounts";

    String? dailyCountsData = prefs.getString(dailyCountsKey);

    Map<String, int> dailyCounts = dailyCountsData != null ? Map<String, int>.from(json.decode(dailyCountsData)) : {};

    double dailyNicotineTarget = threshold \* \_nicotine; // Calcola la soglia giornaliera

    if (registrationDate != null) {

      DateTime startDate = registrationDate!;

      data = [];

      DateTime now = DateTime.now();

      int totalCigarettes = 0;

      double nicotineSmokedToday = 0.0;

      //int daysToGenerate = \_cigarettesPerDay\*7;

      int daysToGenerate = now.difference(startDate).inDays + 1;

      for (int i = 0; i < daysToGenerate; i++) {

        DateTime currentDate = startDate.add(Duration(days: i));

        String dateKey = "${widget.accountName}\_cigarettes\_${currentDate.year}${currentDate.month}${currentDate.day}";

        double cigarettes = dailyCounts[dateKey]?.toDouble() ?? 0.0;

        //totalCigarettes += cigarettes.toInt();

        //nicotineSmokedToday += totalCigarettes \* \_nicotine;

        //print('nicotineSmokedToday ${nicotineSmokedToday}');

        //print('nicotine ${\_nicotine}');

        //print('total cigarettes ${totalCigarettes}');

        data.add(NicotineLevel(date: currentDate, level: cigarettes));

      }

      final cigaretteCounter = Provider.of<CigaretteCounter>(context, listen: false);

      data.add(NicotineLevel(date: now, level: cigaretteCounter.cigarettesSmokedToday.toDouble()));

      totalCigarettes += cigaretteCounter.cigarettesSmokedToday;

      //nicotineSmokedToday = totalCigarettes \* \_nicotine;

      //double cigarettesToday = cigaretteCounter.nicotineSmokedToday.toDouble();

      //nicotineSmokedToday = totalCigarettes \* \_nicotine;

      //print('nicotineSmokedToday ${nicotineSmokedToday}');

      //print('prova: ${cigaretteCounter.nicotineSmokedToday}');

      print('prova2: ${cigaretteCounter.cigarettesSmokedToday}');  //OK VA BENE

      nicotineSmokedToday = cigaretteCounter.cigarettesSmokedToday \* \_nicotine;

      int futureDays = totalCigarettes \* 7;

      for (int i = 1; i <= futureDays; i++) {

        DateTime futureDate = now.add(Duration(days: i));

        data.add(NicotineLevel(date: futureDate, level: 0.0));

      }

      setState(() {

        // Questa variabile sarà usata per mostrare il conteggio nel widget

        this.nicotineSmokedToday = nicotineSmokedToday;

        this.dailyNicotineTarget = dailyNicotineTarget;

      });

GENERATE CHART DATA

 Future<void> \_generateChartData(Map<String, dynamic> users) async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String dailyCountsKey = "${widget.accountName}\_dailyCounts";

    String? dailyCountsData = prefs.getString(dailyCountsKey);

    Map<String, int> dailyCounts = dailyCountsData != null ? Map<String, int>.from(json.decode(dailyCountsData)) : {};

    double dailyNicotineTarget = threshold \* \_nicotine; // Calcola la soglia giornaliera

    if (registrationDate != null) {

      DateTime startDate = registrationDate!;

      data = [];

      DateTime now = DateTime.now();

      int totalCigarettes = 0;

      double nicotineSmokedToday = 0.0;

      //int daysToGenerate = \_cigarettesPerDay\*7;

      int daysToGenerate = now.difference(startDate).inDays + 1;

      for (int i = 0; i < daysToGenerate; i++) {

        DateTime currentDate = startDate.add(Duration(days: i));

        String dateKey = "${widget.accountName}\_cigarettes\_${currentDate.year}${currentDate.month}${currentDate.day}";

        double cigarettes = dailyCounts[dateKey]?.toDouble() ?? 0.0;

        //totalCigarettes += cigarettes.toInt();

        //nicotineSmokedToday += totalCigarettes \* \_nicotine;

        //print('nicotineSmokedToday ${nicotineSmokedToday}');

        //print('nicotine ${\_nicotine}');

        //print('total cigarettes ${totalCigarettes}');

        data.add(NicotineLevel(date: currentDate, level: cigarettes));

      }

      final cigaretteCounter = Provider.of<CigaretteCounter>(context, listen: false);

      data.add(NicotineLevel(date: now, level: cigaretteCounter.cigarettesSmokedToday.toDouble()));

      totalCigarettes += cigaretteCounter.cigarettesSmokedToday;

      //nicotineSmokedToday = totalCigarettes \* \_nicotine;

      //double cigarettesToday = cigaretteCounter.nicotineSmokedToday.toDouble();

      //nicotineSmokedToday = totalCigarettes \* \_nicotine;

      //print('nicotineSmokedToday ${nicotineSmokedToday}');

      //print('prova: ${cigaretteCounter.nicotineSmokedToday}');

      print('prova2: ${cigaretteCounter.cigarettesSmokedToday}');  //OK VA BENE

      nicotineSmokedToday = totalCigarettes \* \_nicotine;

      int futureDays = totalCigarettes \* 7;

      for (int i = 1; i <= futureDays; i++) {

        DateTime futureDate = now.add(Duration(days: i));

        data.add(NicotineLevel(date: futureDate, level: 0.0));

      }

      setState(() {

        // Questa variabile sarà usata per mostrare il conteggio nel widget

        this.nicotineSmokedToday = nicotineSmokedToday;

        this.dailyNicotineTarget = dailyNicotineTarget;

      });

    }

  }

NICOTINE CHART IN PLOT\_CREATION:

  NicotineChart(

    this.seriesList, {

    this.animate = false,

    required this.registrationDate,

    required this.cigarettesPerDay,

    required this.nicotineSmokedToday,

    required this.dailyNicotineTarget,

  });

  @override

  Widget build(BuildContext context) {

    // Calcola la data di inizio e di fine basate sulla data di registrazione e sul numero di sigarette

    //DateTime fixedStartDate = registrationDate;

    //DateTime fixedEndDate = registrationDate.add(Duration(days: cigarettesPerDay \* 7 + 1));

    DateTime fixedStartDate =

        registrationDate.subtract(Duration(days: 1)); // Un giorno prima

    DateTime fixedEndDate = registrationDate

        .add(Duration(days: cigarettesPerDay \* 7 + 2)); // Un giorno dopo

    return charts.TimeSeriesChart(

      seriesList,

      animate: animate,

      dateTimeFactory: const charts.LocalDateTimeFactory(),

      domainAxis: charts.DateTimeAxisSpec(

        tickFormatterSpec: charts.AutoDateTimeTickFormatterSpec(

          day: charts.TimeFormatterSpec(

            format: 'dd', // Formatta i giorni sull'asse X

            transitionFormat: 'dd MMM',

          ),

        ),

        tickProviderSpec: charts.DayTickProviderSpec(increments: [1]), // Incrementi giorno per giorno

        renderSpec: charts.SmallTickRendererSpec(

          labelStyle: charts.TextStyleSpec(

            fontSize: 12,

            color: charts.MaterialPalette.black,

          ),

          lineStyle: charts.LineStyleSpec(

            color: charts.MaterialPalette.black,

          ),

          axisLineStyle: charts.LineStyleSpec(

            color: charts.MaterialPalette.black,

          ),

        ),

        // Imposta l'intervallo dell'asse X al periodo di registrazione

        viewport: charts.DateTimeExtents(

          start: fixedStartDate,

          end: fixedEndDate,

        ),

      ),

      defaultRenderer: charts.BarRendererConfig<DateTime>(

        groupingType: charts.BarGroupingType.grouped, // Barre raggruppate per ogni giorno

        cornerStrategy: const charts.ConstCornerStrategy(20),

      ),

      primaryMeasureAxis: charts.NumericAxisSpec(

        tickProviderSpec: charts.BasicNumericTickProviderSpec(desiredTickCount: 10),

      ),

    );

  }

  // Crea dati di esempio

  static List<charts.Series<NicotineLevel, DateTime>> createSampleData(List<NicotineLevel> data) {

    return [

      charts.Series<NicotineLevel, DateTime>(

        id: 'Nicotine Level',

        colorFn: (\_, \_\_) => charts.MaterialPalette.blue.shadeDefault,

        domainFn: (NicotineLevel levels, \_) => DateTime(levels.date.year, levels.date.month, levels.date.day), // Ignora le ore

        measureFn: (NicotineLevel levels, \_) => levels.level,

        data: data,

      )

    ];

  }

}

PROVA A VEDERE FILE VECCHI: SIA QUELLO CON IL PICCO DI NICOTINA CON LA CURVA CHE DECRESCE, SIA QUELLO IN CUI C’E’ IL GRAFICO A BARRE. VEDI SE C’E’ LA VERSIONE FUNZIONANTE DEI CONTATORI (dovrebbe essere quella caricata di github

import 'dart:math';

import 'package:flutter/material.dart';

import 'package:shared\_preferences/shared\_preferences.dart';

import 'dart:convert';

import 'package:provider/provider.dart';

import 'cigarette\_counter.dart';

import 'package:progetto/charts/plot\_creation.dart';

class Plots extends StatefulWidget {

  final String accountName;

  Plots({required this.accountName});

  @override

  \_PlotsState createState() => \_PlotsState();

}

class \_PlotsState extends State<Plots> {

  DateTime? registrationDate;

  List<NicotineLevel> data = [];

  List<HourlyNicotineLevel> hourlyData = [];

  int \_cigarettesPerDay = 0;

  int threshold = 0;

  bool isLoading = true;

  @override

  void initState() {

    super.initState();

    \_loadRegistrationData();

  }

  Future<void> \_loadRegistrationData() async {

    try {

      SharedPreferences prefs = await SharedPreferences.getInstance();

      String? usersData = prefs.getString('users');

      Map<String, dynamic> users = usersData != null ? json.decode(usersData) : {};

      final cigaretteCounter = Provider.of<CigaretteCounter>(context, listen:false);

      if (users.containsKey(widget.accountName)) {

        var userProfile = users[widget.accountName];

        String? dateStr = userProfile['registrationDate'];

        \_cigarettesPerDay = userProfile['CigarettesPerDay'] ?? 0;

        threshold = \_cigarettesPerDay; //threshold initialization

        if (dateStr != null) {

          registrationDate = DateTime.parse(dateStr);

        } else {

          registrationDate = DateTime.now();

        }

        // Calcola quanti giorni sono passati dalla registrazione

        if (registrationDate != null) {

          int daysSinceRegistration = DateTime.now().difference(registrationDate!).inDays;

          // Decrementa la soglia di 1 per ogni 7 giorni passati

          threshold -= (daysSinceRegistration ~/ 7);

          if (threshold < 0) threshold = 0; // La soglia non può andare sotto 0

        }

        await \_generateChartData(users);

        await \_generateHourlyData();

      } else {

        print("User not found: ${widget.accountName}");

      }

    } catch (e) {

      print("Error loading registration data: $e");

    } finally {

      setState(() {

        isLoading = false;

      });

    }

  }

  Future<void> \_generateChartData(Map<String, dynamic> users) async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    String dailyCountsKey = "${widget.accountName}\_dailyCounts";

    String? dailyCountsData = prefs.getString(dailyCountsKey);

    Map<String, int> dailyCounts = dailyCountsData != null ? Map<String, int>.from(json.decode(dailyCountsData)) : {};

    if (registrationDate != null) {

      DateTime startDate = registrationDate!;

      data = [];

      DateTime now = DateTime.now();

      int totalCigarettes = 0;

      //int daysToGenerate = \_cigarettesPerDay\*7;

      int daysToGenerate = now.difference(startDate).inDays + 1;

      for (int i = 0; i < daysToGenerate; i++) {

        DateTime currentDate = startDate.add(Duration(days: i));

        String dateKey = "${widget.accountName}\_cigarettes\_${currentDate.year}${currentDate.month}${currentDate.day}";

        double cigarettes = dailyCounts[dateKey]?.toDouble() ?? 0.0;

        totalCigarettes += cigarettes.toInt();

        data.add(NicotineLevel(date: currentDate, level: cigarettes));

      }

      final cigaretteCounter = Provider.of<CigaretteCounter>(context, listen: false);

      data.add(NicotineLevel(date: now, level: cigaretteCounter.cigarettesSmokedToday.toDouble()));

      totalCigarettes += cigaretteCounter.cigarettesSmokedToday;

      int futureDays = totalCigarettes \* 7;

      for (int i = 1; i <= futureDays; i++) {

        DateTime futureDate = now.add(Duration(days: i));

        data.add(NicotineLevel(date: futureDate, level: 0.0));

      }

      setState(() {});

    }

  }

  Future<void> \_generateHourlyData() async {

    SharedPreferences prefs = await SharedPreferences.getInstance();

    DateTime now = DateTime.now();

    DateTime startOfDay = DateTime(now.year, now.month, now.day);

    hourlyData = [];

    // Ottieni il provider del contatore delle sigarette

    final cigaretteCounter = Provider.of<CigaretteCounter>(context, listen: false);

    // Recupera i dati orari delle sigarette e della nicotina

    //CONSIDERA L'IDEA DI METTERE List<int> e List<double>, ma devi modificare tutto il resto in pratica

    int hourlyCigarettesSmoked = cigaretteCounter.hourlyCigarettesSmoked;

    double hourlyNicotine = cigaretteCounter.hourlyNicotine;

    // Mantieni un registro cumulativo delle curve

    Map<DateTime, double> cumulativeNicotineLevels = {};

    // Ottieni il tipo di sigaretta e il livello di nicotina salvati per l'utente corrente

    //String? usersData = prefs.getString('users');

    //Map<String, dynamic> users = usersData != null ? json.decode(usersData) : {};

    //double nicotinePerCigarette = 1.0; // Valore predefinito nel caso non ci siano dati

    //if (users.containsKey(widget.accountName)) {

    //  var userProfile = users[widget.accountName];

    //  nicotinePerCigarette = userProfile['Nicotine'] ?? 1.0;

    //}

    Set<String> keys = prefs.getKeys();

    // Raccogli tutte le sigarette fumate dal salvataggio

    for (String key in keys) {

      if (key.startsWith("${widget.accountName}\_cigarette\_")) {

        DateTime timestamp = DateTime.parse(key.split('\_')[2]);

        DateTime hour = DateTime(timestamp.year, timestamp.month, timestamp.day, timestamp.hour);

        // Aggiungi la curva di nicotina per questa sigaretta al registro cumulativo

        for (int i = 0; i < 24; i++) {

          DateTime currentHour = hour.add(Duration(hours: i));

          double nicotineLevel = hourlyNicotine / pow(2, i \* 60 / 90); // Effetto nel tempo

          if (cumulativeNicotineLevels.containsKey(currentHour)) {

            cumulativeNicotineLevels[currentHour] = cumulativeNicotineLevels[currentHour]! + nicotineLevel;

          } else {

            cumulativeNicotineLevels[currentHour] = nicotineLevel;

          }

        }

      }

    }

    // Aggiungi l'ultima sigaretta appena fumata

    //final cigaretteCounter = Provider.of<CigaretteCounter>(context, listen: false);

    if (cigaretteCounter.hourlyCigarettesSmoked > 0) {

      DateTime lastCigaretteTime = now;

      DateTime hour = DateTime(lastCigaretteTime.year, lastCigaretteTime.month, lastCigaretteTime.day, lastCigaretteTime.hour);

      // Aggiorna solo le ore successive alla nuova sigaretta

      for (int i = 0; i < 24; i++) {

        DateTime currentHour = hour.add(Duration(hours: i));

        double nicotineLevel = hourlyNicotine / pow(2, i \* 60 / 90);

        nicotineLevel \*= cigaretteCounter.hourlyCigarettesSmoked;

        if (cumulativeNicotineLevels.containsKey(currentHour)) {

          cumulativeNicotineLevels[currentHour] = cumulativeNicotineLevels[currentHour]! + nicotineLevel;

        } else {

          cumulativeNicotineLevels[currentHour] = nicotineLevel;

        }

      }

    }

    // Ora converte cumulativeNicotineLevels in hourlyData

    for (int i = 0; i < 24; i++) {

      DateTime hour = startOfDay.add(Duration(hours: i));

      double nicotineLevel = cumulativeNicotineLevels[hour] ?? 0.0;

      hourlyData.add(HourlyNicotineLevel(time: hour, level: nicotineLevel));

    }

    setState(() {});

    //PROVA PER VALIDAZIONE- IN CASO TOGLIERE

    for (var hourly in hourlyData) {

      print('Ora: ${hourly.time}, Livello di nicotina: ${hourly.level}');

    }

  }

  @override

  Widget build(BuildContext context) {

    final cigaretteCounter = Provider.of<CigaretteCounter>(context);

    DateTime now = DateTime.now();

    return Scaffold(

      appBar: AppBar(title: Text('Plots')),

      body: isLoading

          ? Center(child: CircularProgressIndicator())

          : Padding(

              padding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 8.0),

              child: Column(

                children: <Widget>[

                  Padding(

                    padding: const EdgeInsets.all(16.0),

                    child: Row(

                      mainAxisAlignment: MainAxisAlignment.start,

                      children: [

                        Text(

                          'Cigarettes smoked today: ${cigaretteCounter.cigarettesSmokedToday}/$threshold',

                          style: TextStyle(fontSize: 20),

                        ),

                      ],

                    ),

                  ),

                  Container(

                    height: MediaQuery.of(context).size.height \* 0.3,

                    child: SingleChildScrollView(

                      scrollDirection: Axis.horizontal,

                      child: Container(

                        width: (\_cigarettesPerDay \* 7 \* 50.0),

                        child: NicotineChart(

                          NicotineChart.createSampleData(data),

                          animate: true,

                          registrationDate: registrationDate!,

                          cigarettesPerDay: \_cigarettesPerDay,

                        ),

                      ),

                    ),

                  ),

                  Padding(

                    padding: const EdgeInsets.symmetric(horizontal: 16.0, vertical: 8.0),

                    child: Align(

                      alignment: Alignment.centerLeft,

                      child: Text(

                        'Slide horizontally to view more data',

                        style: TextStyle(fontSize: 16),

                      ),

                    ),

                  ),

                  SizedBox(height: 20), // Spazio tra i grafici

                  Padding(

                    padding: const EdgeInsets.all(8.0),

                    child: Text(

                      '${now.day.toString().padLeft(2, '0')}/${now.month.toString().padLeft(2, '0')}/${now.year}',

                      style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold),

                    ),

                  ),

                  Container(

                    height: MediaQuery.of(context).size.height \* 0.3, // Altezza ridotta

                    child: HourlyNicotineChart(

                      HourlyNicotineChart.createSampleData(hourlyData),

                      animate: true,

                    ),

                  ),

                ],

              ),

            ),

    );

  }

}

GENERATE HOURLY DATA AGGIORNATO CHE FUNZIONA

Future<void> \_generateHourlyData(Map<String, dynamic> users) async {

  SharedPreferences prefs = await SharedPreferences.getInstance();

  String hourlyCountsKey = "${widget.accountName}\_hourlyCounts";

  String? hourlyCountsData = prefs.getString(hourlyCountsKey);

  Map<String, int> hourlyCounts = hourlyCountsData != null

      ? Map<String, int>.from(json.decode(hourlyCountsData))

      : {};

  DateTime now = DateTime.now();

  DateTime startOfDay = DateTime(now.year, now.month, now.day); //hourly chart refers to today's date

  hourlyData = [];

  //int hoursToGenerate = now.difference(startOfDay).inHours + 1;

  int hoursToGenerate = 24;

  //print('hours to generate: $hoursToGenerate');

  //print('start of day: $startOfDay');

  DateTime roundedHour= DateTime(now.year, now.month, now.day, now.hour);

  //print('roundedHour: $roundedHour');

  //print('now: $now');

  //int totalCigarettes  = 0;

  int cigarettesSmokedThisHour = 0;

  double nicotineSmokedThisHour = 0.0;

  for (int i = 0; i <= hoursToGenerate; i++) {

    DateTime currentHour = startOfDay.add(Duration(hours: i));

    String hourlyKey = "${widget.accountName}\_hourly\_cigarettes\_${currentHour.year}${currentHour.month}${currentHour.day}${currentHour.hour}";

    ///////SBAGLIATO!!! DA' SEMPRE 0///////////

    double cigarettes = hourlyCounts[hourlyKey]?.toDouble() ?? 0.0;

    //totalCigarettes += cigarettes.toInt();

    //print('cigaretteS: $cigarettes');

    hourlyData.add(HourlyNicotineLevel(time: currentHour, level: cigarettes));

    // Arrotonda l'ora

    //DateTime roundedHour = DateTime(currentHour.year, currentHour.month, currentHour.day, currentHour.hour);

    // Aggiungi o aggiorna il dato orario

    //int existingIndex = hourlyData.indexWhere((entry) => entry.time == roundedHour);

    //if (existingIndex != -1) {

    //  hourlyData[existingIndex].level += cigarettes;

    //} else {

    //  hourlyData.add(HourlyNicotineLevel(time: roundedHour, level: cigarettes));

    //}

    if (currentHour.hour == now.hour &&

        currentHour.day == now.day &&

        currentHour.month == now.month &&

        currentHour.year == now.year) {

      nicotineSmokedThisHour += cigarettes;

    }

  }

  // Aggiungi i dati orari dell'ora corrente

  final nicotineCounter = Provider.of<CigaretteCounter>(context, listen: false);

  hourlyData.add(HourlyNicotineLevel(time: now, level: nicotineCounter.hourlyNicotine));

  nicotineSmokedThisHour += nicotineCounter.hourlyNicotine.toDouble();

  cigarettesSmokedThisHour += nicotineCounter.hourlyCigarettesSmoked;

  //print('cigarettes smoked this hour: $cigarettesSmokedThisHour');

  // Aggiungi dati futuri per le ore seguenti

  DateTime tomorrow = DateTime(now.year, now.month, now.day +1);

  //print('tomorrow: $tomorrow');

  //////////////SISTEMA FUTURE HOURS////////////

  int futureHours = (tomorrow.difference(roundedHour)).inHours; // Visualizza anche ore future

  //print('fotureHours: $futureHours');

  for (int i = 1; i <= futureHours; i++) {

    DateTime futureHour = now.add(Duration(hours: i));

    futureHour = DateTime(futureHour.year, futureHour.month, futureHour.day, futureHour.hour);

    //print('future hours: $futureHour');

    hourlyData.add(HourlyNicotineLevel(time: futureHour, level: 0.0));

  }

  // Aggiorna lo stato con i dati orari

  setState(() {

    this.nicotineSmokedThisHour = nicotineSmokedThisHour;

  });

}