```
// README
// FocusPro - Pro-level Flutter Focus & Study App
// Files included in this single document:
// 1) pubspec.yaml (dependencies)
// 2) lib/main.dart (complete app)
// How to run:
// 1. Create a new Flutter project: `flutter create focus_pro`
// 2. Replace `pubspec.yaml` and `lib/main.dart` with the contents below.
// 3. Run `flutter pub get` then `flutter run` on a device/emulator.

--- pubspec.yaml ---
name: focus_pro
description: A pro-level Focus & Study Flutter app (single-file demo)
publish_to: 'none'
version: 1.0.0+1
environment:
  sdk: '>=2.18.0 <3.0.0'

dependencies:
  flutter:
    sdk: flutter
  shared_preferences: ^2.1.1
  flutter_local_notifications: ^9.6.1
  audioplayers: ^2.1.1
  percent_indicator: ^4.2.2

flutter:
  uses-material-design: true

--- lib/main.dart ---
import 'dart:async';
```

```
import 'package:flutter/material.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:flutter_local_notifications/flutter_local_notifications.dart';
import 'package:audioplayers/audioplayers.dart';
import 'package:percent_indicator/percent_indicator.dart';

void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  final prefs = await SharedPreferences.getInstance();
  runApp(FocusProApp(prefs: prefs));
}

class FocusProApp extends StatelessWidget {
  final SharedPreferences prefs;
  FocusProApp({required this.prefs});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'FocusPro',
      theme: ThemeData(
        primarySwatch: Colors.indigo,
        scaffoldBackgroundColor: Colors.grey[50],
      ),
      home: HomePage(prefs: prefs),
    );
  }
}

class HomePage extends StatefulWidget {
```

```
  final SharedPreferences prefs;
  HomePage({required this.prefs});
  @override
  _HomePageState createState() => _HomePageState();
}

enum TimerState { stopped, running, paused }

class _HomePageState extends State<HomePage> with WidgetsBindingObserver {
  int studyMinutes = 25;
  int breakMinutes = 5;
  late Duration remaining;
  TimerState timerState = TimerState.stopped;
  Timer? _timer;
  late FlutterLocalNotificationsPlugin flutterLocalNotificationsPlugin;
  final AudioPlayer audioPlayer = AudioPlayer();
  int streak = 0;
  int totalMinutesToday = 0;

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addObserver(this);
    remaining = Duration(minutes: studyMinutes);
    _initNotifications();
    _loadStats();
  }

  Future<void> _initNotifications() async {
    flutterLocalNotificationsPlugin = FlutterLocalNotificationsPlugin();
    const AndroidInitializationSettings initializationSettingsAndroid =
```

```
    AndroidInitializationSettings('@mipmap/ic_launcher');
  final InitializationSettings initializationSettings = InitializationSettings(
    android: initializationSettingsAndroid,
  );
  await flutterLocalNotificationsPlugin.initialize(initializationSettings);
}

Future<void> _showNotification(String title, String body) async {
  const AndroidNotificationDetails androidPlatformChannelSpecifics =
      AndroidNotificationDetails('focus_channel', 'Focus Alerts',
          channelDescription: 'Notifications for FocusPro',
          importance: Importance.max,
          priority: Priority.high,
          ticker: 'ticker');
  const NotificationDetails platformChannelSpecifics =
      NotificationDetails(android: androidPlatformChannelSpecifics);
  await flutterLocalNotificationsPlugin.show(
      0, title, body, platformChannelSpecifics);
}

Future<void> _loadStats() async {
  setState(() {
    streak = widget.prefs.getInt('streak') ?? 0;
    totalMinutesToday = widget.prefs.getInt('minutes_today') ?? 0;
  });
}

Future<void> _saveStats() async {
  await widget.prefs.setInt('streak', streak);
  await widget.prefs.setInt('minutes_today', totalMinutesToday);
}
```

```
void _startTimer({required Duration duration}) {
  _timer?.cancel();
  setState(() {
    remaining = duration;
    timerState = TimerState.running;
  });
  _timer = Timer.periodic(Duration(seconds: 1), (timer) {
    if (remaining.inSeconds <= 1) {
      timer.cancel();
      _onTimerComplete();
    } else {
      setState(() {
        remaining = Duration(seconds: remaining.inSeconds - 1);
      });
    }
  });
}

void _pauseTimer() {
  _timer?.cancel();
  setState(() {
    timerState = TimerState.paused;
  });
}

void _stopTimer() {
  _timer?.cancel();
  setState(() {
    timerState = TimerState.stopped;
    remaining = Duration(minutes: studyMinutes);
```

```
  });
}

Future<void> _onTimerComplete() async {
  await _playSound();
  await _showNotification('FocusPro', 'Session complete! Take a break.');

  // update stats only if it was a study session
  setState(() {
    streak += 1;
    totalMinutesToday += studyMinutes;
  });
  await _saveStats();

  // auto-start break timer
  _startTimer(duration: Duration(minutes: breakMinutes));
}

Future<void> _playSound() async {
  try {
    await audioPlayer.play(AssetSource('note.mp3'));
  } catch (e) {
    // asset might not exist in demo, ignore
  }
}

double get _progress {
  final total = Duration(minutes: studyMinutes).inSeconds;
  final done = total - remaining.inSeconds;
  return (done / total).clamp(0.0, 1.0);
}
```

```
@override
void dispose() {
  WidgetsBinding.instance.removeObserver(this);
  _timer?.cancel();
  audioPlayer.dispose();
  super.dispose();
}

// When app goes to background, keep timer running (simple demo)
@override
void didChangeAppLifecycleState(AppLifecycleState state) {
  if (state == AppLifecycleState.paused) {
    // For production: save timestamp to compute elapsed time later
  }
}

@override
Widget build(BuildContext context) {
  final minutes = remaining.inMinutes.remainder(60).toString().padLeft(2, '0');
  final seconds = (remaining.inSeconds.remainder(60)).toString().padLeft(2, '0');

  return Scaffold(
    appBar: AppBar(
      title: Text('FocusPro'),
      actions: [
        IconButton(
          icon: Icon(Icons.bar_chart),
          onPressed: () {
            Navigator.of(context).push(MaterialPageRoute(
              builder: (_) => StatsPage(streak: streak, minutesToday: totalMinutesToday)));
```

```
            })
        ],
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            SizedBox(height: 12),
            Text('Study: ${studyMinutes}m • Break: ${breakMinutes}m', style: TextStyle(fontSize: 16)),
            SizedBox(height: 20),
            CircularPercentIndicator(
              radius: 160.0,
              lineWidth: 14.0,
              percent: _progress,
              center: Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Text('$minutes:$seconds', style: TextStyle(fontSize: 36, fontWeight: FontWeight.bold)),
                  SizedBox(height: 6),
                  Text(timerState == TimerState.running ? 'Studying' : timerState == TimerState.paused ? 'Paused' : 'Ready')
                ],
              ),
              progressColor: Colors.indigo,
            ),
            SizedBox(height: 24),

            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
```

```
    ElevatedButton(
     onPressed: () {
      if (timerState == TimerState.stopped || timerState == TimerState.paused) {
       _startTimer(duration: Duration(minutes: studyMinutes));
      } else {
       _pauseTimer();
      }
     },
     child: Text(timerState == TimerState.running ? 'Pause' : 'Start'),
    ),
    SizedBox(width: 12),
    OutlinedButton(onPressed: _stopTimer, child: Text('Stop')),
    SizedBox(width: 12),
    ElevatedButton(
      onPressed: () async {
       final res = await _showSettingsDialog();
       if (res != null) {
        setState(() {
         studyMinutes = res['study'];
         breakMinutes = res['break'];
         if (timerState == TimerState.stopped) remaining = Duration(minutes: studyMinutes);
        });
       }
      },
      child: Text('Settings'))
   ],
  ),

  SizedBox(height: 24),
  Divider(),
  SizedBox(height: 8),
```

```
          FeatureTile(
            icon: Icons.lock_clock,
            title: 'Lock Mode (Android only - guide)',
            subtitle:
                'Guide: implement Android Accessibility Service or Device Admin for real app blocking. Not included in demo for permissions reasons.',
          ),
          FeatureTile(
            icon: Icons.volume_up,
            title: 'Calm Sounds',
            subtitle: 'Play background ambient sounds while studying (add audio assets in assets/).',
          ),
          FeatureTile(
            icon: Icons.emoji_events,
            title: 'Streaks & Rewards',
            subtitle: 'Streaks increment on completion. Stored locally using shared_preferences.',
          ),
        ],
      ),
    ),
  );
}

Future<Map<String, int>?> _showSettingsDialog() async {
  final studyController = TextEditingController(text: studyMinutes.toString());
  final breakController = TextEditingController(text: breakMinutes.toString());
  return showDialog<Map<String, int>>(
    context: context,
    builder: (context) {
      return AlertDialog(
        title: Text('Settings'),
```

```
          content: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              TextField(controller: studyController, keyboardType: TextInputType.number, decoration: InputDecoration(labelText: 'Study minutes')),
              TextField(controller: breakController, keyboardType: TextInputType.number, decoration: InputDecoration(labelText: 'Break minutes')),
            ],
          ),
          actions: [
            TextButton(onPressed: () => Navigator.of(context).pop(), child: Text('Cancel')),
            TextButton(
              onPressed: () {
                final s = int.tryParse(studyController.text) ?? studyMinutes;
                final b = int.tryParse(breakController.text) ?? breakMinutes;
                Navigator.of(context).pop({'study': s, 'break': b});
              },
              child: Text('Save'))
          ],
        );
      });
  }
}

class FeatureTile extends StatelessWidget {
  final IconData icon;
  final String title;
  final String subtitle;
  FeatureTile({required this.icon, required this.title, required this.subtitle});
  @override
  Widget build(BuildContext context) {
    return ListTile(
```

```
      leading: Icon(icon, size: 28),
      title: Text(title, style: TextStyle(fontWeight: FontWeight.bold)),
      subtitle: Text(subtitle),
    );
  }
}

class StatsPage extends StatelessWidget {
  final int streak;
  final int minutesToday;
  StatsPage({required this.streak, required this.minutesToday});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Stats')),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text('Current Streak', style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
            SizedBox(height: 8),
            Text('$streak days'),
            SizedBox(height: 16),
            Text('Minutes today', style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
            SizedBox(height: 8),
            Text('$minutesToday minutes'),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () async {
```

```
            // In a full app: export CSV or sync to cloud
            ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Export / Sync coming in pro build')));
          },
          child: Text('Export') )
        ],
      ),
    ),
  );
 }
}
```

--- NOTES & NEXT STEPS ---

1) App Blocking / Lock Mode: For a real lock feature on Android you must implement a native Android Accessibility Service or device owner APIs; this requires writing platform-specific code (Kotlin/Java) and asking for sensitive permissions. For iOS it's not allowed the same way — you can provide guidance and shortcuts but not full blocking.

2) Assets: add an audio file `assets/note.mp3` and configure assets: in pubspec.yaml add:

```
# assets:
#   - assets/note.mp3
```

3) To make it production-ready: add tests, background service for timers, secure cloud sync (Firebase), onboarding UX, and analytics.

4) Want this converted to React Native or a complete multi-file repo with native app-lock implementation? I can create that too.

// End of document