Flutter Background Services

Learn how to run tasks in the background, fetch data, and handle notifications for fully functional mobile apps.

Website reference: haas.dev | https://dev-roast-app.vercel.app

---

Introduction

Many mobile apps need to perform tasks while the user is not actively using the app. Background services allow your Flutter app to **fetch data, send notifications, or update content** even when running in the background, improving functionality and user engagement.

Step 1: Understanding Background Services

Types of background tasks in Flutter:

1. **Background Fetch** – Periodic data updates.
2. **Push Notifications Handling** – Triggered by FCM or local notifications.
3. **Scheduled Tasks / Cron Jobs** – Tasks run at specified intervals.
4. **Isolates** – For heavy computations without blocking the main UI thread.

Step 2: Setting Up Background Fetch

Use `background_fetch` plugin:

```
dependencies:
  background_fetch: ^1.0.0
```

Initialize in `main.dart`:

```
void main() {
  runApp(MyApp());
  BackgroundFetch.configure(
    BackgroundFetchConfig(
      minimumFetchInterval: 15,
      stopOnTerminate: false,
      enableHeadless: true,
    ), (String taskId) async {
      print("Background fetch executed: $taskId");
```

```
      BackgroundFetch.finish(taskId);
    });
}
```

- `minimumFetchInterval`: Time between background fetches (in minutes).
- `stopOnTerminate`: Keep running after app is killed.
- `enableHeadless`: Run even if app is not active.

Step 3: Using Isolates for Heavy Tasks

- Isolates allow heavy computations without blocking the main UI.
- Example: Image processing or data parsing in the background.

```
import 'dart:isolate';

void heavyTask(SendPort sendPort) {
  // Heavy computation
  sendPort.send('Task completed');
}

void startIsolate() async {
  final receivePort = ReceivePort();
  await Isolate.spawn(heavyTask, receivePort.sendPort);
  receivePort.listen((message) {
    print(message);
  });
}
```

Step 4: Background Push Notifications

Use **Firebase Cloud Messaging (FCM)**:

- Configure FCM in Firebase console.
- Add `firebase_messaging` plugin:

```
dependencies:
  firebase_messaging: ^14.0.0
```

- Handle background messages:

```
FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgroundHandler);

Future<void> _firebaseMessagingBackgroundHandler(RemoteMessage message) async {
  print("Handling a background message: ${message.messageId}");
}
```

Step 5: Scheduled Tasks & Cron Jobs

Use `android_alarm_manager_plus` for scheduled tasks:

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await AndroidAlarmManager.initialize();
  await AndroidAlarmManager.periodic(
    const Duration(hours: 1),  // Run every hour
    0,  // Unique ID
    printTask,
  );
}

void printTask() {
  print("Scheduled task executed");
}
```

- iOS has limitations; consider push notifications for scheduling.

Step 6: Practical Exercise

Objective: Create a Flutter app that fetches data in the background and notifies the user.

1. Setup **background fetch** every 30 minutes.
2. Use **FCM** to trigger notifications when new data is available.
3. Implement an **isolate** to process fetched data without blocking UI.

Test by closing the app and verifying notifications and background tasks still execute.

Step 7: Key Takeaways

- Background services improve app usability and engagement.

- Use **background_fetch** for periodic tasks.
- Use **isolates** for heavy computations.
- Handle **push notifications** with FCM for real-time updates.
- Use **scheduled tasks** for recurring background operations.

Visit **haas.dev** for more resources and guides.
https://dev-roast-app.vercel.app
