실시간 매칭 관련 기능 조사

1. Flutter에서의 역할

1. **라이프사이클 상태 관리**:
   - Flutter에서는 **`WidgetsBindingObserver`*를 사용하여 앱의 라이프사이클 상태를 감지할 수 있다.
   - 앱이 **포그라운드(`active`)**, **백그라운드(`background`)**, 또는 종료 상태로 전환될 때 이를 감지해 처리 가능하다.
2. **주기적인 상태 업데이트**:
   - `Timer` 또는 백그라운드 작업 플러그인을 활용해, 주기적으로 현재 앱 상태를 Spring Boot 서버로 전송할 수 있다.
   - 예를 들어, 5분 단위로 POST 요청을 통해 사용자 상태(`active`, `background`)를 서버에 업데이트하여 매칭 가능 여부를 유지한다.
3. **백그라운드 상태에서 WebSocket 유지**:
   - Flutter는 기본적으로 백그라운드 상태로 전환되면 WebSocket 연결이 끊어질 수 있지만, **백그라운드 작업 플러그인**을 사용하면 이를 방지할 수 있다.
   - Flutter에서 사용 가능한 주요 플러그인은 다음과 같다:
     - `WorkManager`:
       - 주기적인 작업 실행 가능.
       - 백그라운드 상태에서도 WebSocket 연결을 주기적으로 유지하거나, 끊어진 경우 재연결 시도를 처리할 수 있다.
       - Android의 `JobScheduler`와 iOS의 `BGTaskScheduler`를 기반으로 동작한다.
     - `background_fetch`:
       - 특정 주기로 클라이언트에서 작업을 실행할 수 있으며, WebSocket 연결 재유지를 위한 로직을 작성할 수 있다.
4. **FCM(Firebase Cloud Messaging) 활용**:
   - Flutter에서 **Firebase Cloud Messaging(FCM)**을 활용하면 백그라운드 상태나 앱 종료 상태에서도 알림(Push Notification)을 받을 수 있다.
   - 실시간 매칭 알림을 서버로부터 받아 사용자에게 표시 가능하다.

2. Spring Boot에서의 역할

1. **사용자 상태 관리**:
   - Flutter 클라이언트가 주기적으로 전송하는 상태 정보를 저장하여 사용자 상태(`active`, `background`, `inactive`)를 관리한다.
   - 일정 시간 동안 상태 업데이트가 없는 경우, 해당 사용자를 `inactive`로 처리하여 비활성 연결을 정리한다.
2. **실시간 통신(WebSocket)**:
   - WebSocket을 사용하여 클라이언트와 서버 간 실시간 양방향 연결을 유지한다.
   - 매칭 상태, 결과 등을 즉시 전달하며, WebSocket 연결을 통해 안정적으로 실시간 통신을 지원한다.
3. **비활성 연결 종료 및 재연결 처리**:
   - 서버는 Ping/Pong 메시지를 사용해 연결 상태를 감지하고, 일정 시간 동안 응답이 없으면 연결을 종료한다.
   - 연결이 끊어진 경우, 클라이언트에서 자동으로 재연결 시도를 통해 통신을 복구할 수 있다.

---

3. 백그라운드에서 WebSocket 유지 가능성

Flutter에서는 기본적으로 WebSocket 연결이 백그라운드 상태에서 끊길 수 있지만, 플러그인을 통해 이를 방지할 수 있다.

WorkManager를 이용한 WebSocket 유지

1. **주기적인 WebSocket 연결 작업**:
   - WorkManager는 **15분 단위로 백그라운드에서 작업을 실행**할 수 있으며, WebSocket 연결 유지나 재연결 작업을 처리할 수 있다.
2. **Flutter에서 WorkManager 설정 예제**:

```
dart
import 'dart:io';
import 'package:workmanager/workmanager.dart';

void backgroundTaskDispatcher() {
  Workmanager().executeTask((task, inputData) async {
```

```
    try {
      // WebSocket 연결
      final webSocket = await WebSocket.connect('ws://your-
server-url/ws/match');
      print("백그라운드 WebSocket 연결 성공");

      // 메시지 수신 대기
      webSocket.listen((data) {
        print("백그라운드 메시지 수신: $data");
      });

      return Future.value(true); // 작업 성공 반환
    } catch (e) {
      print("백그라운드 WebSocket 연결 실패: $e");
      return Future.value(false); // 작업 실패 반환
    }
  });
}

void main() {
  WidgetsFlutterBinding.ensureInitialized();

  // WorkManager 초기화
  Workmanager().initialize(backgroundTaskDispatcher, isInDe
bugMode: true);

  // 백그라운드 작업 등록
  Workmanager().registerPeriodicTask(
    "websocketTask",
    "백그라운드 WebSocket 작업",
    frequency: Duration(minutes: 15),
  );

  runApp(MyApp());
}
```

WorkManager 장점

- 백그라운드 상태에서 주기적으로 작업을 실행해 WebSocket 연결 유지.
- 필요 시 재연결 로직을 추가해 안정적인 통신 보장.

---

4. 주요 고려사항

1. **WebSocket과 FCM의 조합**:
   - 실시간 매칭 로직은 WebSocket을 통해 구현하고, 백그라운드 및 종료 상태에서는 FCM을 사용해 푸시 알림을 전달한다.
2. **배터리 효율**:
   - WebSocket 연결을 유지하면 배터리 소모가 증가할 수 있으므로, 필요한 경우 WorkManager를 통해 주기적인 연결을 설정하여 최적화한다.
3. **Flutter 백그라운드 제한**:
   - iOS의 경우 백그라운드 작업 시간 제한이 있으므로, `BGTaskScheduler`를 통해 작업을 처리하거나, 일정 주기로 FCM 알림을 대체할 수 있다.

결론

Flutter와 Spring Boot를 활용한 실시간 매칭 기능은 다음과 같은 방식으로 구현 가능하다:

1. Flutter는 **라이프사이클 상태를 감지**해 현재 상태를 Spring Boot로 주기적으로 업데이트.
2. 백그라운드 상태에서도 **WorkManager**와 같은 플러그인을 활용해 WebSocket 연결을 유지하거나, 끊어진 경우 재연결 시도.
3. Spring Boot는 WebSocket을 통해 실시간 매칭 결과를 전달하며, FCM으로 알림을 보완해 사용자가 매칭 상태를 안정적으로 확인할 수 있도록 지원.