router.g.dart

소스 코드

```dart
// GENERATED CODE - DO NOT MODIFY BY HAND

part of 'router.dart';

// **************************************************************************
// RiverpodGenerator
// **************************************************************************

String _$routerHash() => r'cbb381475f4dbc08db20ceaa319e7d2583e5a47b';

/// See also [router].
@ProviderFor(router)
final routerProvider = Provider<GoRouter>.internal(
  router,
  name: r'routerProvider',
  debugGetCreateSourceHash: const bool.fromEnvironment('dart.vm.product')
      ? null
      : _$routerHash,
  dependencies: null,
  allTransitiveDependencies: null,
);

@Deprecated('Will be removed in 3.0. Use Ref instead')
// ignore: unused_element
typedef RouterRef = ProviderRef<GoRouter>;
// ignore_for_file: type=lint
// ignore_for_file: subtype_of_sealed_class, invalid_use_of_internal_member,
invalid_use_of_visible_for_testing_member, deprecated_member_use_from_same_package

```

개요

`router.g.dart` 파일은 Riverpod 코드 생성기(`riverpod_generator`)에 의해 자동으로 생성되는 소스 파일이다.
이 파일은 `router.dart`에 정의된 `router` 프로바이더를 Riverpod 런타임에 등록하기 위한 메타데이터와 팩토리 코드를 포함합니다.
해당 파일은 자동 생성되기 때문에 수동으로 수정하면 안 된다. 빌드 과정(`build_runner`)에서 언제든지 덮어씌워진다.

---

part 지시자

```dart
part of 'router.dart';
```

이 지시자는 `router.dart`와 이 파일을 하나의 라이브러리로 결합한다.
두 파일은 동일한 라이브러리 스코프를 공유하며, private 멤버에도 접근 가능하다.

해시 함수

```
String _$routerHash() => r'cbb381475f4dbc08db20ceaa319e7d2583e5a47b';
```

이 해시는 생성된 프로바이더의 고유 식별자 역할을 한다.
디버깅, 변경 감지, Hot Reload 반영 등 내부적인 최적화 목적에 사용된다.
릴리즈 모드(`dart.vm.product=true`)에서는 성능 최적화를 위해 사용되지 않는다.

---

routerProvider 정의

```
final routerProvider = Provider<GoRouter>.internal(
  router,
  name: r'routerProvider',
  debugGetCreateSourceHash: const bool.fromEnvironment('dart.vm.product')
      ? null
      : _$routerHash,
  dependencies: null,
  allTransitiveDependencies: null,
);
```

구성 요소 설명

- **Provider.internal**
  - Riverpod의 내부 생성자를 사용하여 `router` 함수를 프로바이더로 등록한다.
  - 일반적인 생성자 대신 내부 생성자를 사용하는 이유는 빌드 생성 코드와의 최적화 때문이다.
- **router**
  - `router.dart`에서 정의된 실제 GoRouter 생성 함수.
- **name**
  - 프로바이더의 식별자 이름. DevTools, 로깅 등에서 사용된다.
- **debugGetCreateSourceHash**
  - 디버그 모드에서 해시 함수를 사용하여 소스 변경 감지.
  - 릴리즈 모드에서는 null로 비활성화됨.
- **dependencies / allTransitiveDependencies**
  - 이 프로바이더가 의존하는 다른 프로바이더 목록.
  - 현재 router 프로바이더는 다른 프로바이더에 의존하지 않으므로 null.

RouterRef 타입 별칭 (Deprecated)

```
typedef RouterRef = ProviderRef<GoRouter>;
```

이 타입은 과거 버전에서 사용되던 Ref 타입 별칭이다.
Riverpod 3.0에서 제거될 예정이며, `Ref`를 직접 사용하는 방식으로 전환해야 한다.

ignore 규칙

파일 하단의 ignore 문들은 자동 생성 코드에서 발생하는 linter 경고를 무시하기 위한 설정이다.

```
// ignore_for_file: type=lint
// ignore_for_file: subtype_of_sealed_class, invalid_use_of_internal_member, ...
```

빌드 생성기 코드 특성상 내부 API 사용이 불가피하여 이러한 ignore 설정이 포함된다.

📌 왜 자동 생성 파일(router.g.dart)이 프로젝트에 그대로 보이는가?

1) Dart/Flutter의 코드 생성 방식이 "파일을 직접 생성하는 방식"이기 때문

Flutter/Riverpod의 코드 생성 시스템(`build_runner`)은
다른 언어(예: Java/Kotlin: annotation processor, Swift: build-time dynamic generation)처럼
"빌드 후 내부 메모리에만 존재하는 코드"를 만드는 구조가 아니다.

→ **Dart는 "source-gen" 방식**을 사용한다.
→ 빌드 도중 **실제 .dart 파일을 생성하고 프로젝트 폴더에 저장하는 구조**이다.

즉,
자동 생성 파일을 실제로 눈에 보여야
Dart analyzer, IDE, 컴파일러가 그 파일을 가져다 사용할 수 있다.

📌 2) IDE(Analyzer)가 자동 생성 코드를 읽어야 타입 검사·자동완성·Jump-to-definition이 가능함

Riverpod, Freezed, JsonSerializable 등이 `.g.dart` 파일을 생성하는 이유가 바로 이것이다.

예를 들어:

- Provider 타입 분석
- ref.watch 자동완성
- 오류 검사
- 타입 추론
- 네비게이션(Jump to Definition)

이 기능들은 **IDE가 파일을 직접 읽어야만** 작동한다.

그래서 숨기지 않는다.

📌 3) '숨기지 않는 대신, 자동 생성 코드임을 명확히 표시'하는 표준 방식

자동 생성 코드는 반드시 **다음 헤더**를 갖는다:

```
// GENERATED CODE - DO NOT MODIFY BY HAND
```

즉,

- *"사용자는 절대 수정하지 말라"* 는 강한 경고는 제공하지만,
- *소스 트리에서는 숨기지 않는다.*

→ 이것은 Dart 생태계 전체의 표준이다.

📌 4) Git 커밋에서 제외하는 것도 선택사항

가끔은 `.g.dart` 파일을 Git에 포함하지 않고,

```
.g.dart
.freezed.dart
```

이런 식으로 `.gitignore`에 넣기도 한다.

하지만 공식 권장 방식은 다음과 같다:

✔ **Riverpod, Freezed, JsonSerializable → 일반적으로 Git에 포함한다**

이유:
CI/CD나 빌드 환경에서 build_runner가 항상 돌아가는 것이 아니라면
생성 파일이 없어서 앱이 빌드되지 않는 문제가 발생할 수 있음.

📌 결론 (핵심)

❗ 자동으로 생성되는 파일이지만
❗ Dart/Flutter 생태계는 **"생성된 파일을 프로젝트에 명시적으로 둔다"** 방식이다.
❗ IDE·타입 체크·컴파일러 모두 이 파일을 직접 읽어야 하기 때문이다.

따라서:

- 숨기지 않고 프로젝트에 노출되는 것이 정상
- 수정만 하지 않으면 문제 없음
- 자동 생성기(build_runner)가 매번 책임지고 갱신함