GraphQL

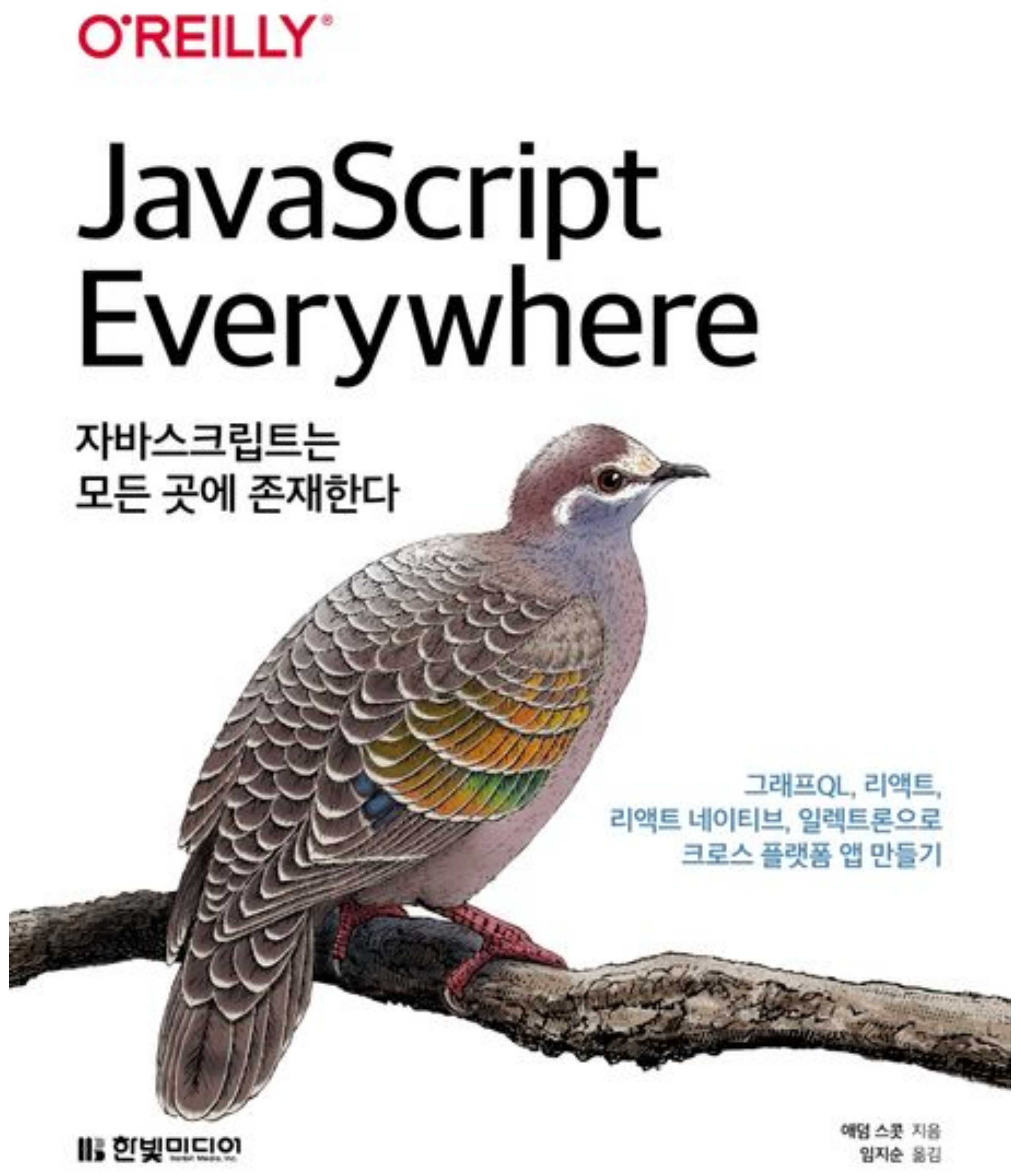
JavaScriptEverywhere

2024.05.10(금) - 이시영

JavascriptEverywhere

자바스크립트는
모든 곳에 존재한다

O'REILLY



O'REILLY

JavaScript Everywhere

자바스크립트는
모든 곳에 존재한다

그래프QL, 리액트,
리액트 네이티브, 일렉트론으로
크로스 플랫폼 앱 만들기

한빛미디어

애덤 스콧 지음
임지순 옮김

목차

전반적인 GraphQL 동작 흐름

기본 구성 요소

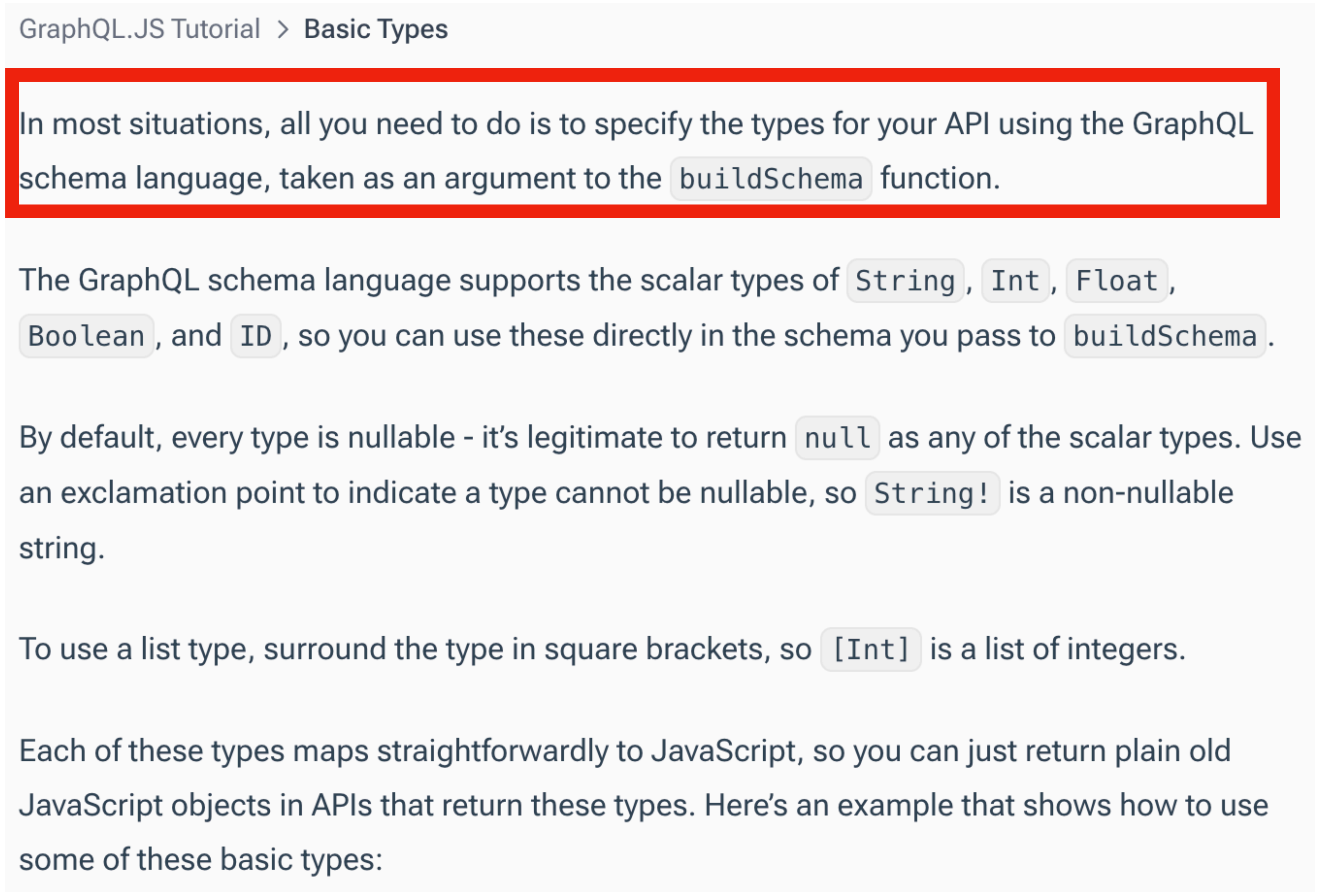
기초 필수 내용 복습

Schema 작성

GraphQL schema, 왜 작성하는걸까

API의 타입을 명시하기 위해

GraphQL schema Language 사용



GraphQL.JS Tutorial > **Basic Types**

In most situations, all you need to do is to specify the types for your API using the GraphQL schema language, taken as an argument to the `buildSchema` function.

The GraphQL schema language supports the scalar types of `String`, `Int`, `Float`, `Boolean`, and `ID`, so you can use these directly in the schema you pass to `buildSchema`.

By default, every type is nullable - it's legitimate to return `null` as any of the scalar types. Use an exclamation point to indicate a type cannot be nullable, so `String!` is a non-nullable string.

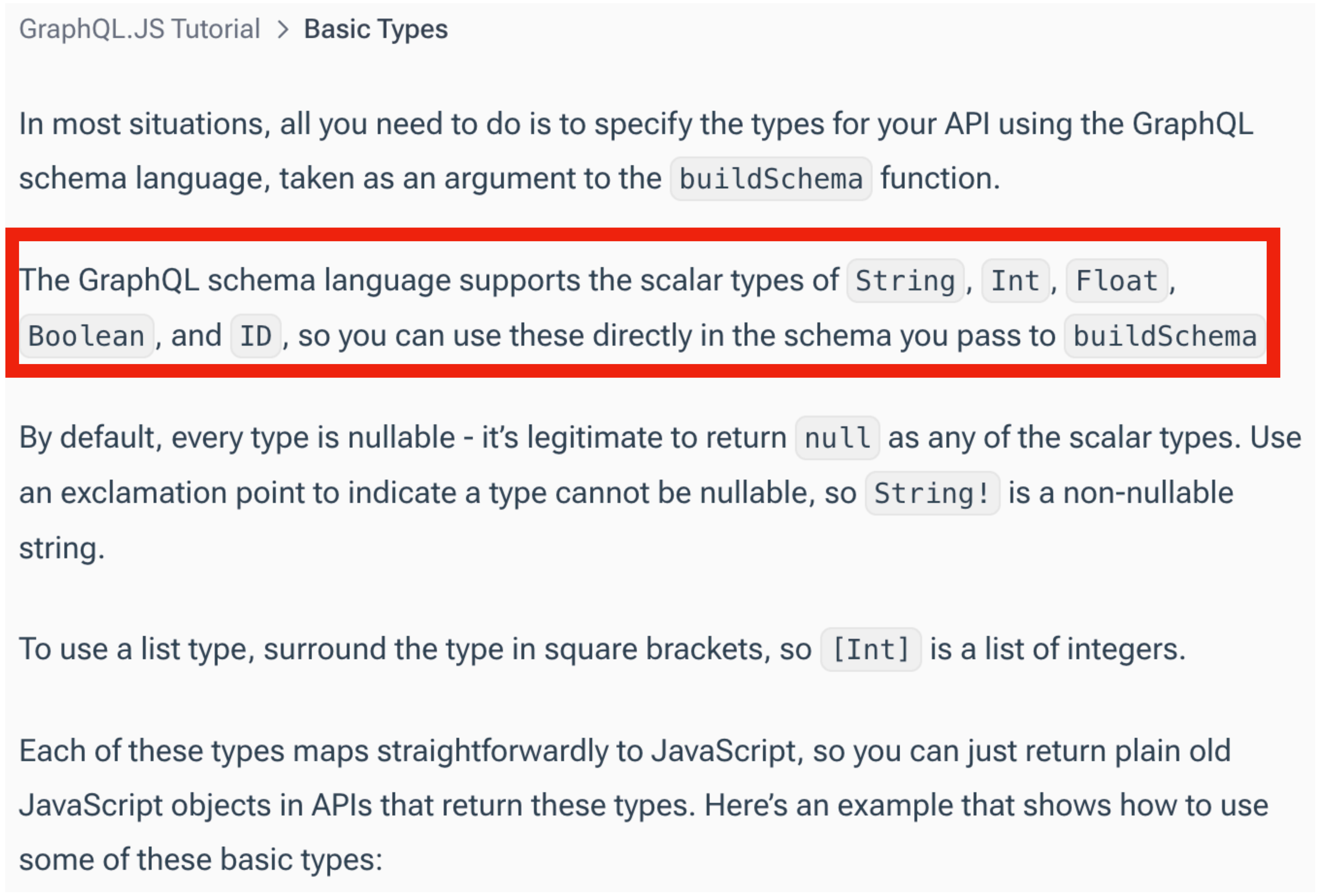
To use a list type, surround the type in square brackets, so `[Int]` is a list of integers.

Each of these types maps straightforwardly to JavaScript, so you can just return plain old JavaScript objects in APIs that return these types. Here's an example that shows how to use some of these basic types:

Schema 작성

Scalar Type

- **String**
- **Int**
- **Float**
- **Boolean**
- **ID**
  - 객체를 다시 요청하거나 cache의 키로써 자주 사용되는 고유 식별자



GraphQL.JS Tutorial > **Basic Types**

In most situations, all you need to do is to specify the types for your API using the GraphQL schema language, taken as an argument to the `buildSchema` function.

The GraphQL schema language supports the scalar types of `String`, `Int`, `Float`, `Boolean`, and `ID`, so you can use these directly in the schema you pass to `buildSchema`

By default, every type is nullable - it's legitimate to return `null` as any of the scalar types. Use an exclamation point to indicate a type cannot be nullable, so `String!` is a non-nullable string.

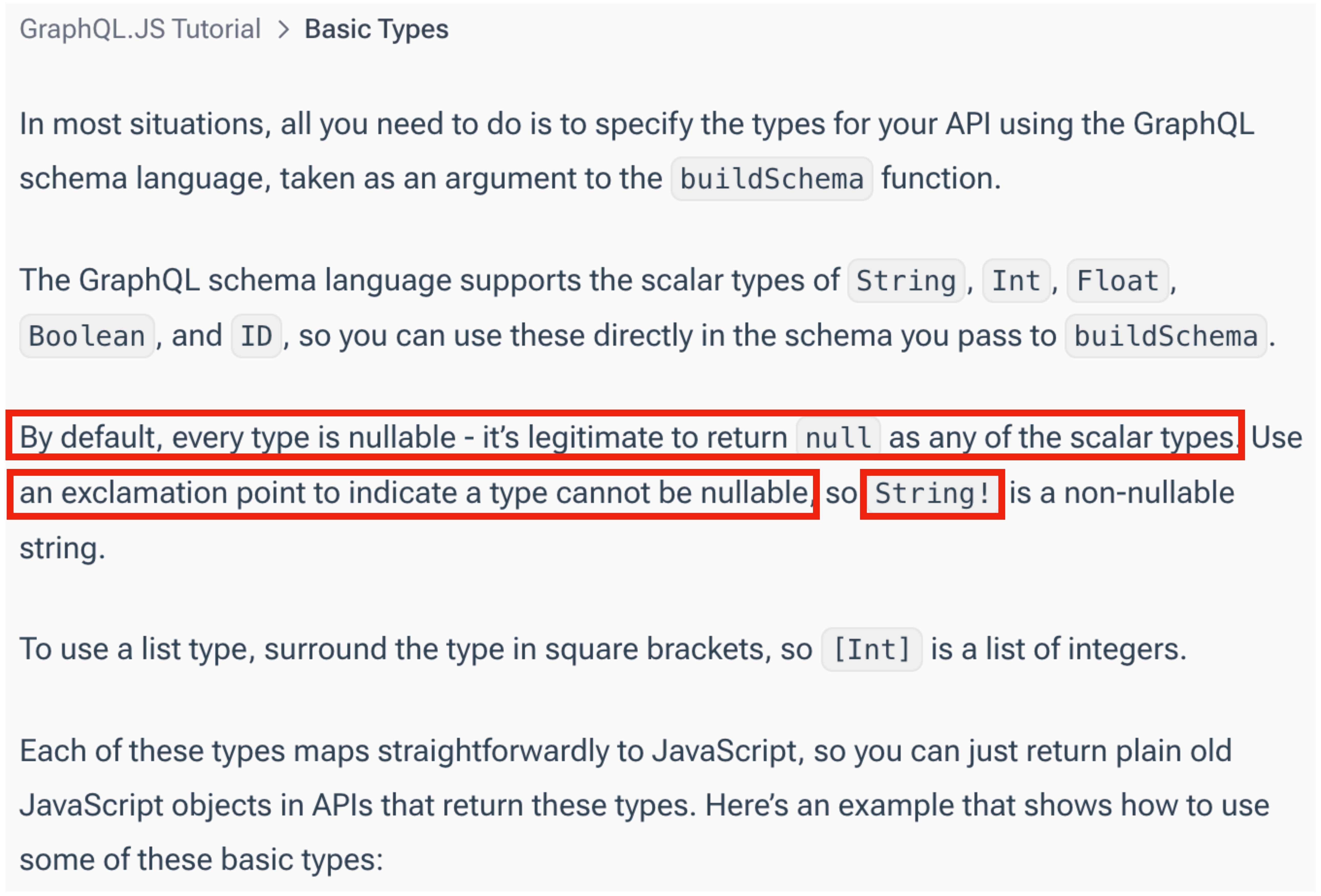
To use a list type, surround the type in square brackets, so `[Int]` is a list of integers.

Each of these types maps straightforwardly to JavaScript, so you can just return plain old JavaScript objects in APIs that return these types. Here's an example that shows how to use some of these basic types:

Schema 작성

Nullable VS Non-Nullable

- 기본적으로,
  모든 타입은 Null로 반환될 수 있음
- 특정 타입은 Null로 반환되지 않기를 원하면, **!**를 사용하여 **Null반환을 방지**



GraphQL.JS Tutorial > **Basic Types**

In most situations, all you need to do is to specify the types for your API using the GraphQL schema language, taken as an argument to the `buildSchema` function.

The GraphQL schema language supports the scalar types of `String`, `Int`, `Float`, `Boolean`, and `ID`, so you can use these directly in the schema you pass to `buildSchema`.

By default, every type is nullable - it's legitimate to return `null` as any of the scalar types. Use an exclamation point to indicate a type cannot be nullable, so `String!` is a non-nullable string.

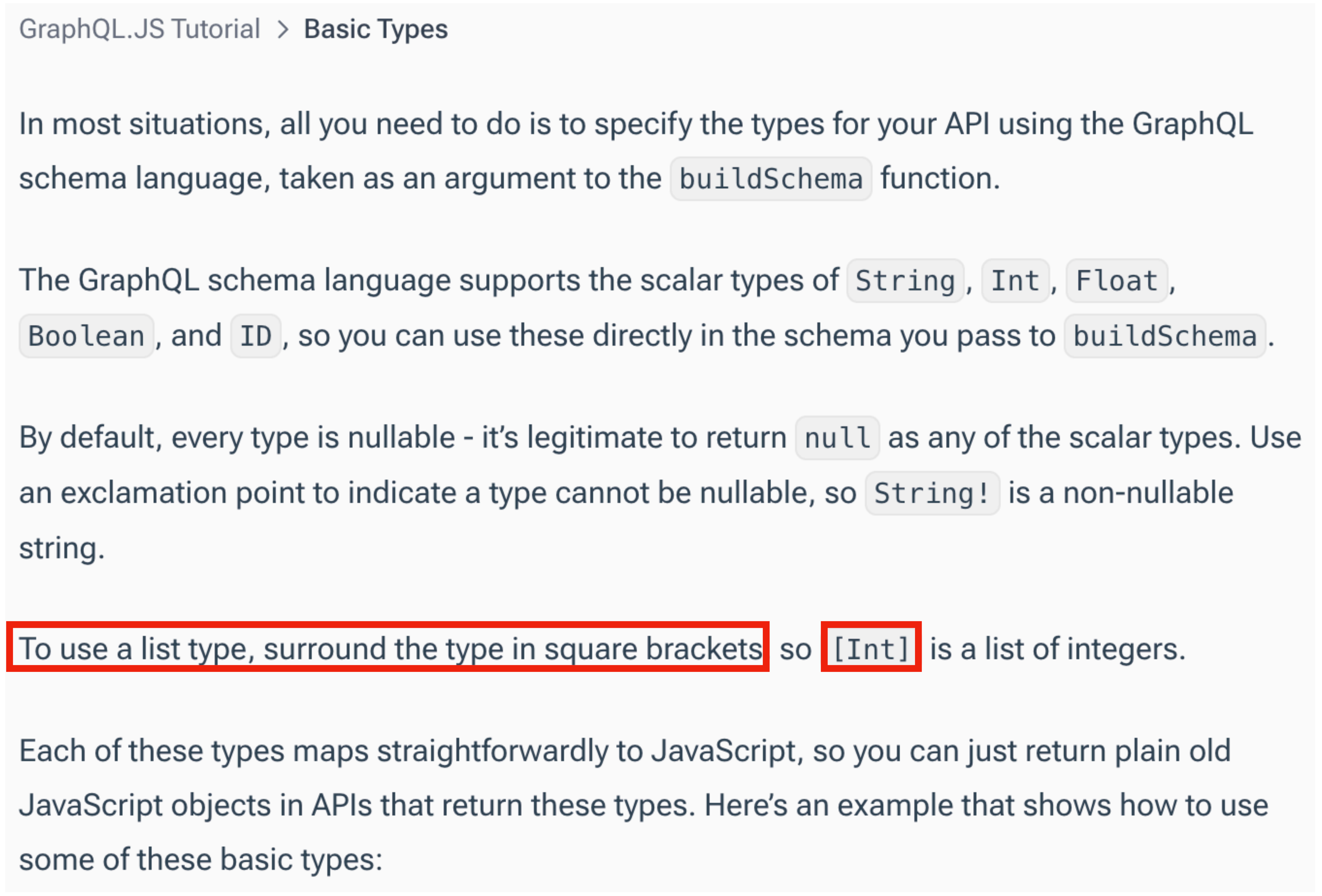
To use a list type, surround the type in square brackets, so `[Int]` is a list of integers.

Each of these types maps straightforwardly to JavaScript, so you can just return plain old JavaScript objects in APIs that return these types. Here's an example that shows how to use some of these basic types:

Schema 작성

List

[]를 사용



GraphQL.JS Tutorial > **Basic Types**

In most situations, all you need to do is to specify the types for your API using the GraphQL schema language, taken as an argument to the `buildSchema` function.

The GraphQL schema language supports the scalar types of `String`, `Int`, `Float`, `Boolean`, and `ID`, so you can use these directly in the schema you pass to `buildSchema`.

By default, every type is nullable - it's legitimate to return `null` as any of the scalar types. Use an exclamation point to indicate a type cannot be nullable, so `String!` is a non-nullable string.

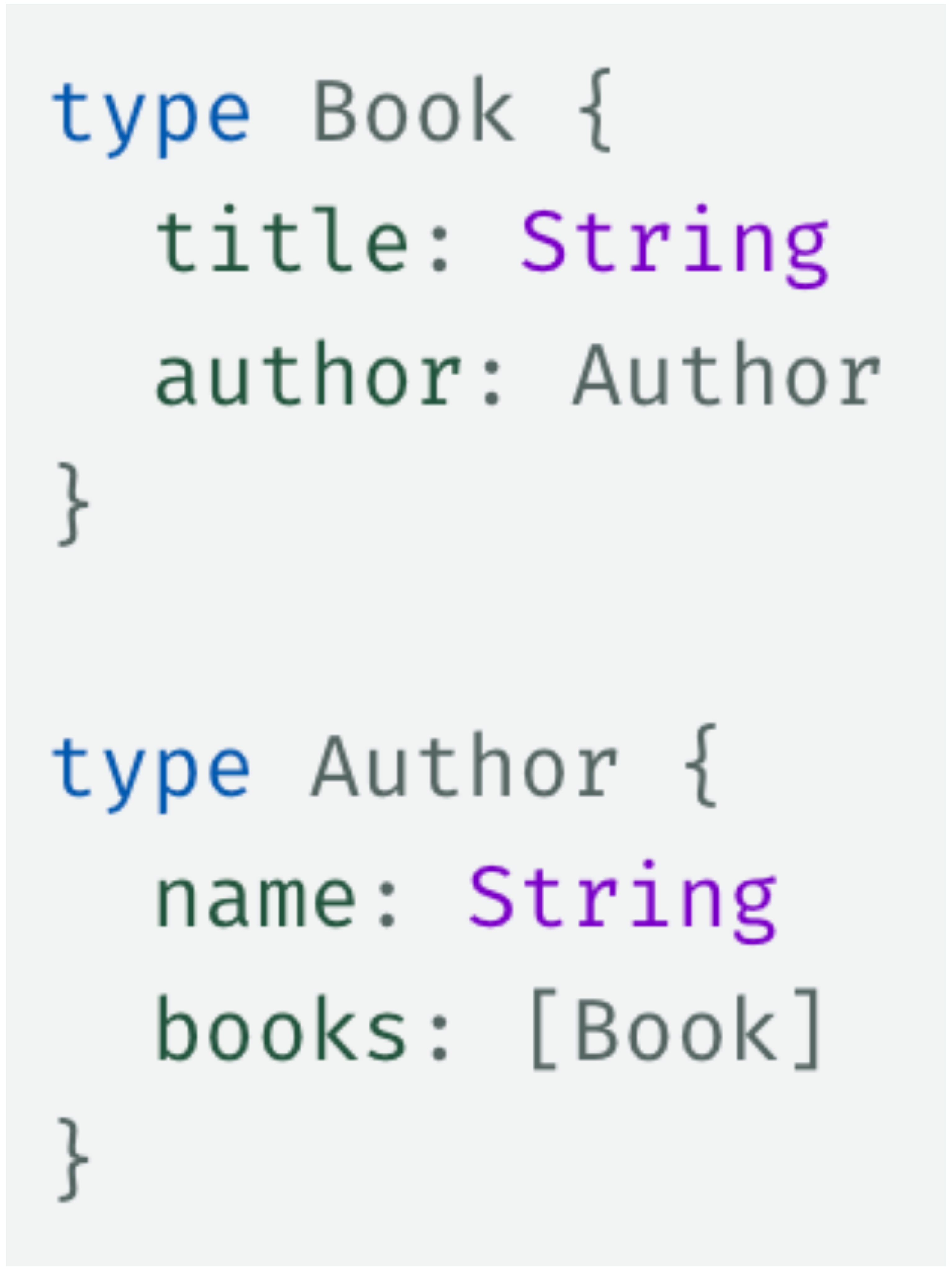
To use a list type, surround the type in square brackets so `[Int]` is a list of integers.

Each of these types maps straightforwardly to JavaScript, so you can just return plain old JavaScript objects in APIs that return these types. Here's an example that shows how to use some of these basic types:

Schema 작성

Object Type

- Type들을 중괄호({})를 통해 **객체형태(key:value 형태)로 만들기 가능**
- **Custom한 Object Type을 Object Type**내에 **특정 키의 반환 타입으로 설정 가능**

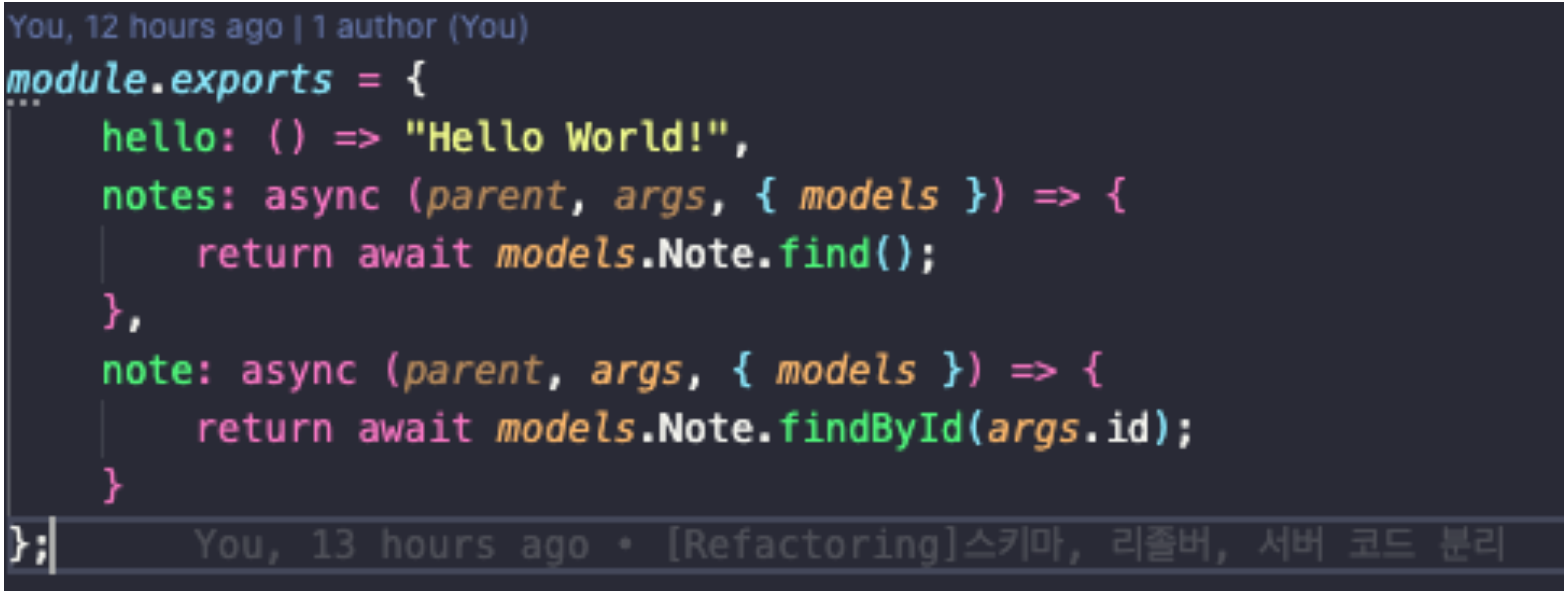


```
type Book {
  title: String
  author: Author
}

type Author {
  name: String
  books: [Book]
}
```

Resolver 작성

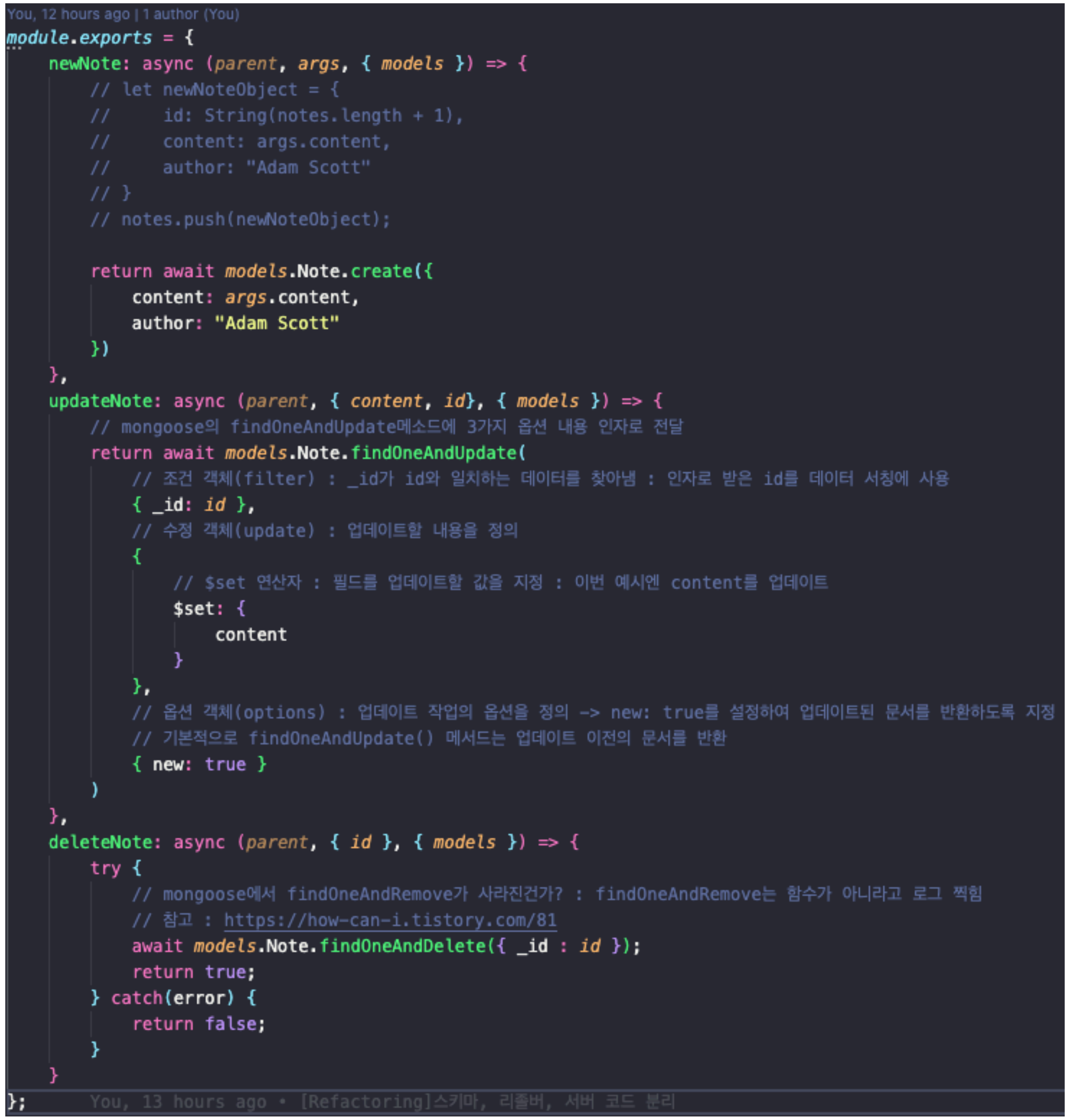
요청 방식1 : Query



```
You, 12 hours ago | 1 author (You)
module.exports = {
    hello: () => "Hello World!",
    notes: async (parent, args, { models }) => {
        return await models.Note.find();
    },
    note: async (parent, args, { models }) => {
        return await models.Note.findById(args.id);
    }
};        You, 13 hours ago • [Refactoring]스키마, 리졸버, 서버 코드 분리
```

Resolver 작성

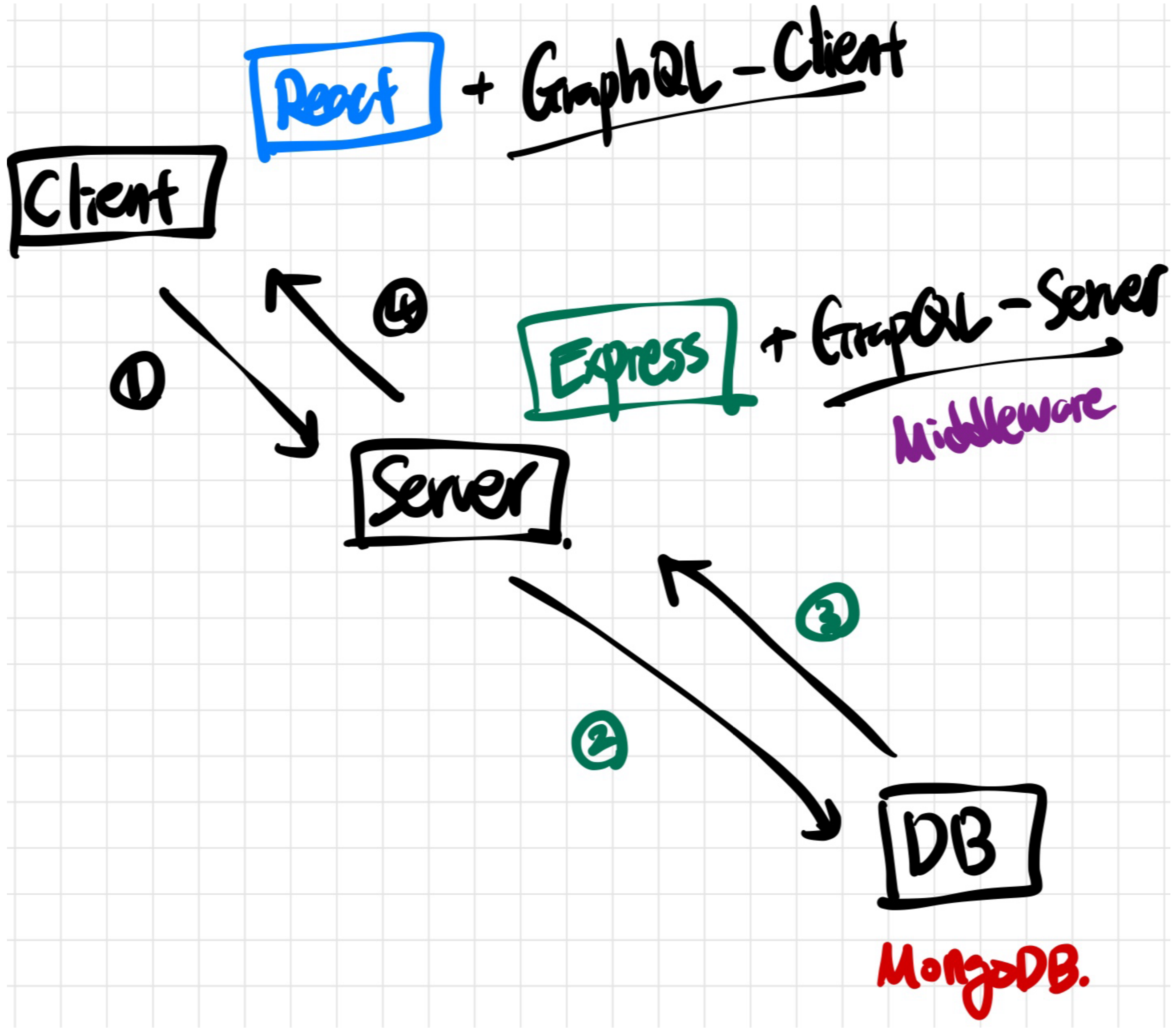
요청 방식2 : Mutation

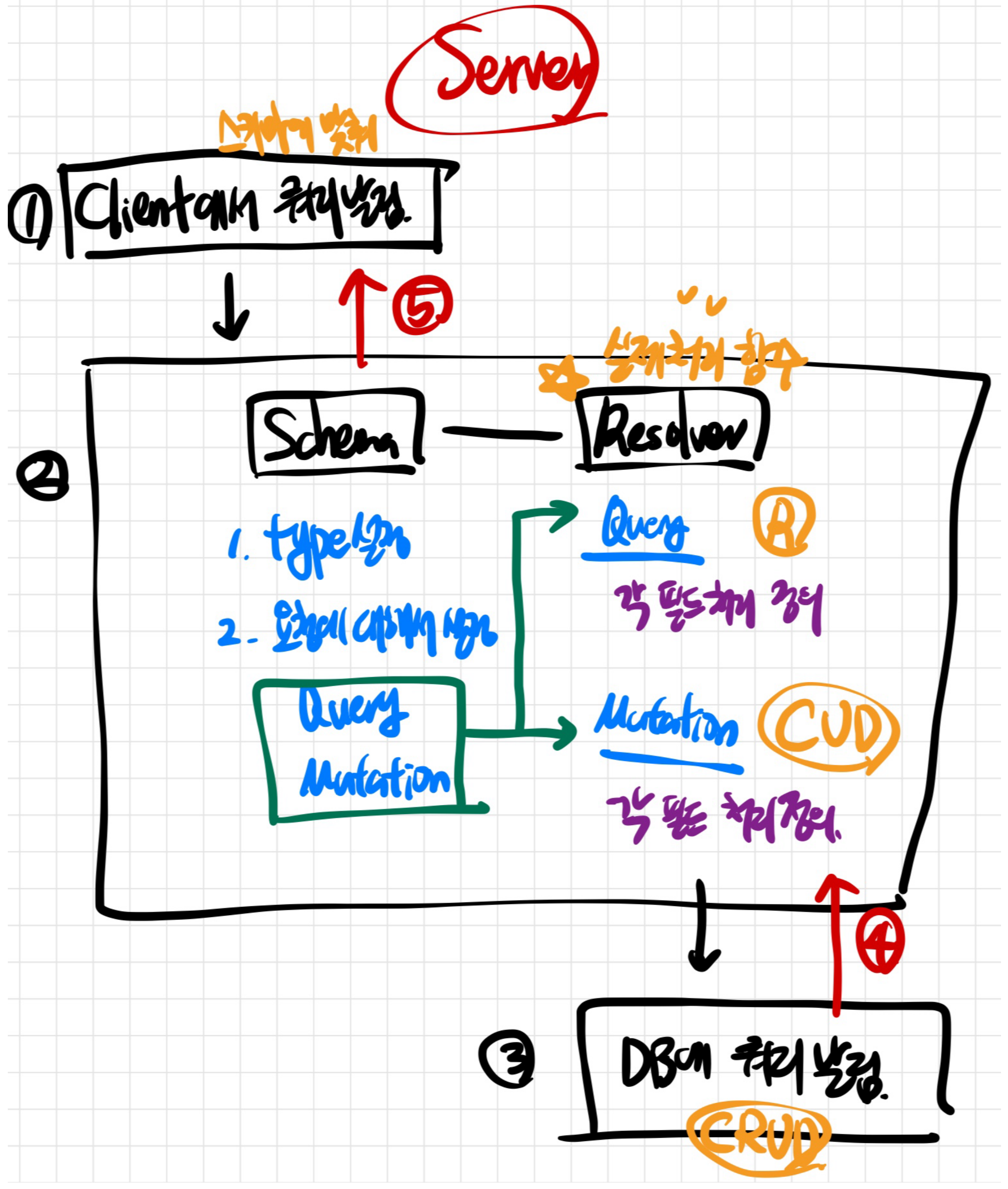


```
You, 12 hours ago | 1 author (You)
module.exports = {
    newNote: async (parent, args, { models }) => {
        // let newNoteObject = {
        //     id: String(notes.length + 1),
        //     content: args.content,
        //     author: "Adam Scott"
        // }
        // notes.push(newNoteObject);

        return await models.Note.create({
            content: args.content,
            author: "Adam Scott"
        })
    },
    updateNote: async (parent, { content, id}, { models }) => {
        // mongoose의 findOneAndUpdate메소드에 3가지 옵션 내용 인자로 전달
        return await models.Note.findOneAndUpdate(
            // 조건 객체(filter) : _id가 id와 일치하는 데이터를 찾아냄 : 인자로 받은 id를 데이터 서칭에 사용
            { _id: id },
            // 수정 객체(update) : 업데이트할 내용을 정의
            {
                // $set 연산자 : 필드를 업데이트할 값을 지정 : 이번 예시엔 content를 업데이트
                $set: {
                    content
                }
            },
            // 옵션 객체(options) : 업데이트 작업의 옵션을 정의 -> new: true를 설정하여 업데이트된 문서를 반환하도록 지정
            // 기본적으로 findOneAndUpdate() 메서드는 업데이트 이전의 문서를 반환
            { new: true }
        )
    },
    deleteNote: async (parent, { id }, { models }) => {
        try {
            // mongoose에서 findOneAndRemove가 사라진건가? : findOneAndRemove는 함수가 아니라고 로그 찍힘
            // 참고 : https://how-can-i.tistory.com/81
            await models.Note.findOneAndDelete({ _id : id });
            return true;
        } catch(error) {
            return false;
        }
    }
};      You, 13 hours ago • [Refactoring]스키마, 리졸버, 서버 코드 분리
```

GraphQL 전체 흐름





잠시 코드 보고 오겠습니다

코드를 통해 파악하는 앱 실행 흐름

마무리

Java - SpringBoot

이 3가지의 공통점

타입 체크

TypeScript

GraphQL

타입

애플리케이션의 사이즈가 커지면서

코드 파악 시간에 피로감을 느끼는 중이라고 봐야겠죠?

(인자값에 어떤 타입이 넘어오는지도 찾아보면서 하는건,,,😮‍💨)

수고하셨습니다

2024.05.10(금) - 이시영