Serving over HTTP

- https://graphql.org/learn/serving-over-http/

HTTP는 어디에나 있기 때문에 GraphQL을 사용할 때 클라이언트-서버 프로토콜에 대한 가장 일반적인 선택입니다. 다음은 HTTP를 통해 작동하도록 GraphQL 서버를 설정하기 위한 몇 가지 지침입니다.[1)]

Web Request Pipeline

대부분의 최신 웹 프레임워크는 요청이 미들웨어 스택(일명 필터/플러그인)을 통해 전달되는 파이프라인 모델을 사용합니다. 요청이 파이프라인을 통해 흐르면 응답으로 검사, 변환, 수정 또는 종료할 수 있습니다. GraphQL은 모든 인증 미들웨어 뒤에 위치해야 HTTP 엔드포인트 핸들러에서와 동일한 세션 및 사용자 정보에 액세스할 수 있습니다.[2)]

URIs, Routes

HTTP는 일반적으로 " 리소스 " 를 핵심 개념으로 사용하는 REST와 연결됩니다. 대조적으로, GraphQL의 개념적 모델은 엔터티 그래프입니다. 결과적으로 GraphQL의 엔터티는 URL로 식별되지 않습니다. 대신 GraphQL 서버는 단일 URL/엔드포인트(일반적으로 `/graphql`)에서 작동하며 주어진 서비스에 대한 모든 GraphQL 요청은 이 엔드포인트로 보내야 합니다.[3)]

HTTP Methods, Headers, and Body

GraphQL HTTP 서버는 HTTP GET 및 POST 메서드를 처리해야 합니다.[4)]

GET request

HTTP GET 요청을 받으면 "query" 쿼리 문자열에 GraphQL 쿼리를 지정해야 합니다. 예를 들어 다음 GraphQL 쿼리를 실행하려는 경우:[5)]

```
{
  me {
    name
  }
}
```

이 요청은 다음과 같이 HTTP GET을 통해 보낼 수 있습니다.[6)]

```
http://myapi/graphql?query={me{name}}
```

쿼리 변수는 변수라는 추가 쿼리 매개변수에서 JSON 인코딩 문자열로 보낼 수 있습니다. 쿼리에 여러 명명된 작업이 포함된 경우 operationName 쿼리 매개변수를 사용하여 실행할 작업을 제어할 수 있습니다.[7)]

POST request

표준 GraphQL POST 요청은 application/json 콘텐츠 유형을 사용해야 하며 다음 형식의 JSON 인코딩 본문을 포함해야 합니다.[8)]

```
{
  "query": "...",
  "operationName": "...",
  "variables": { "myVariable": "someValue", ... }
}
```

operationName 및 변수는 선택적 필드입니다. operationName은 쿼리에 여러 작업이 있는 경우에만 필요합니다.[9)]

위의 내용 외에도 두 가지 추가 사례를 지원하는 것이 좋습니다.[10)]

- “query” 쿼리 문자열 매개변수가 있는 경우(위의 GET 예제에서와 같이) HTTP GET 경우와 동일한 방식으로 구문 분석 및 처리되어야 합니다.[11)]
- “application/graphql” Content-Type 헤더가 있는 경우 HTTP POST 본문 내용을 GraphQL 쿼리 문자열로 처리합니다.[12)]

express-graphql을 사용하는 경우 이미 이러한 동작을 무료로 얻을 수 있습니다.[13)]

Response

쿼리 및 변수를 보낸 방법에 관계없이 응답은 요청 본문에 JSON 형식으로 반환되어야 합니다. 사양에서 언급했듯이 쿼리는 일부 데이터와 일부 오류를 유발할 수 있으며, 이는 다음 형식의 JSON 객체로 반환되어야 합니다.[14)]

```
{
  "data": { ... },
  "errors": [ ... ]
}
```

반환된 오류가 없으면 응답에 “ 오류 ” 필드가 없어야 합니다. 데이터가 반환되지 않으면 GraphQL 사양에 따라 “ 데이터 ” 필드는 실행 중에 오류가 발생하지 않은 경우에만 포함되어야 합니다.[15)]

GraphiQL

GraphiQL은 테스트 및 개발 중에 유용하지만 기본적으로 프로덕션에서는 비활성화되어야 합니다.

express-graphql을 사용하는 경우 NODE_ENV 환경 변수를 기반으로 전환할 수 있습니다.[16)]

```
app.use('/graphql', graphqlHTTP({
  schema: MySessionAwareGraphQLSchema,
  graphiql: process.env.NODE_ENV === 'development',
}));
```

Node

NodeJS를 사용하는 경우 서버 구현 목록을 살펴보는 것이 좋습니다.[17)]

Continue Reading

- Authorization

관련 문서

Plugin Backlinks: 아무 것도 없습니다.

1)
HTTP is the most common choice for client-server protocol when using GraphQL because of its ubiquity. Here are some guidelines for setting up a GraphQL server to operate over HTTP.

2)
Most modern web frameworks use a pipeline model where requests are passed through a stack of middleware (AKA filters/plugins). As the request flows through the pipeline, it can be inspected, transformed, modified, or terminated with a response. GraphQL should be placed after all authentication middleware, so that you have access to the same session and user information you would in your HTTP endpoint handlers.

3)
HTTP is commonly associated with REST, which uses “resources” as its core concept. In contrast, GraphQL's conceptual model is an entity graph. As a result, entities in GraphQL are not identified by URLs. Instead, a GraphQL server operates on a single URL/endpoint, usually /graphql, and all GraphQL requests for a given service should be directed at this endpoint.

4)
GraphQL HTTP 서버는 HTTP GET 및 POST 메서드를 처리해야 합니다.

5)
When receiving an HTTP GET request, the GraphQL query should be specified in the “query” query string. For example, if we wanted to execute the following GraphQL query:

6)
This request could be sent via an HTTP GET like so:

7)
Query variables can be sent as a JSON-encoded string in an additional query parameter called variables. If the query contains several named operations, an operationName query parameter can be used to control which one should be executed.

8)
A standard GraphQL POST request should use the application/json content type, and include a JSON-encoded body of the following form:

9)

operationName and variables are optional fields. operationName is only required if multiple operations are present in the query.

10)

In addition to the above, we recommend supporting two additional cases:

11)

If the “query” query string parameter is present (as in the GET example above), it should be parsed and handled in the same way as the HTTP GET case.

12)

If the “application/graphql” Content-Type header is present, treat the HTTP POST body contents as the GraphQL query string.

13)

If you're using express-graphql, you already get these behaviors for free.

14)

Regardless of the method by which the query and variables were sent, the response should be returned in the body of the request in JSON format. As mentioned in the spec, a query might result in some data and some errors, and those should be returned in a JSON object of the form:

15)

If there were no errors returned, the “errors” field should not be present on the response. If no data is returned, according to the GraphQL spec, the “data” field should only be included if no errors occurred during execution.

16)

GraphiQL is useful during testing and development but should be disabled in production by default. If you are using express-graphql, you can toggle it based on the NODE_ENV environment variable:

17)

If you are using NodeJS, we recommend looking at the list of server implementations.