

The diagram illustrates thread execution within a process. It consists of two main parts: a top section with two boxes and a bottom section with two boxes. The top-left box, labeled 'Main Thread', contains a list of tasks: '1. print' and '2. print(data)'. The top-right box, labeled 'Sub Thread', contains the task '2. get db()'. The bottom-left box is labeled 'Process (python server.py)' and the bottom-right box is labeled 'Process (node server.js)'. Arrows indicate that the threads are part of their respective processes.

Main
Thread
1. print
2. print(data)

Sub Thread
2. get db()

Process
(python server.py)

Process
(node server.js)

<http://localhost:4500/>

Chrome

<http://localhost:4500/>

`http.createServer()`

Hello World

<http://localhost:4500/>

Chrome

Web Server
(html, css,
image)

nginx
apache
openrsty

WAS Server
(Data, Biz Logic)

express
tomcat/jeus
weblogic
django/wsgi

Database

<http://82cook.com/write>

Chrome

요청 파라미터 (GET)

`http://82cook.com/?title=오늘날씨는어때요 &content=블라블라블라`

요청 본문 (POST)

`title = 오늘날씨는 어때요 & content =`

`블라블라블발브블라블라블발브블라블라블발브
블라블라블발브블라블라블발브블라블라블발브
블라블라블발브블라블라블발브블라블라블발브
블라블라블발브블라블라블발브`

요청 본문 (POST)

`{"title": "오늘날씨는어때요?", "content": "블라블라블라"}`

Web Server
(html, css,
image)

nginx
apache
openrsty





1번 게시물

2번 게시물

3번 게시물

4번 게시물

5번 게시물

6번 게시물

7번 게시물

1페이지

```
select * from posts  
limit 5 offset 0
```

2페이지

```
select * from posts  
limit 5 offset 5
```

GET /posts

GET /posts/1

POST /posts → 게시글 생성

PUT /posts/1 → 게시글 수정

PATCH /posts/1 → 게시글 일부 업데이트

DELETE /posts/1 → 게시글이 1번인 자원 삭제

사용자 목록 : /users.

게시글 목록 : /posts

GET /getAllPosts

DELETE /users/123

200 OK 요청 성공

201 자원 생성 성공

204 성공했으나 반환할 데이터가 없다

400 잘못된 요청

401 인증 실패

403 권한이 없다

404 자원을 찾을 수 없다

500 서버에 오류

요청 URI : /posts/1

응답 JSON

```
{“id” : 1, “title”: “테스트 제목”, “content”:”테스트 내용”
```

```
“links” : {“rel”:“self”, “href”: “/posts/1”, “rel”:”comments”, “href”:”/posts/1/comments”
```

```
}
```

페이징, 필터링, 정렬

페이징 : GET /posts?page=2&limit=20

정렬 : GET /posts?sort=createdAt

필터링 : GET /posts?author=지훈

캐싱

Cache-Control : max-age = 3600 (초) -> 1시간동안 캐싱

보안

JWT (JsonWeb Token) → 로컬 인증

OAuth2 를 이용해서 인증 방식 → 애플 로그인, 구글 로그인, 카카톡 로그인,
네이버 로그인

버전 관리

URI 를 통한 버전 관리 : GET /v1/posts /v2/posts

헤더정보를 통한 버전관리 : GET /posts HEADER → Accept:
application/vnd.company.v2+json

공지사항 API

공지사항 목록 : GET /posts

공지사항 상세 : GET /posts/1

공지사항 쓰기 : POST /posts

공지사항 수정 : PUT /posts/1

공지사항 삭제 : DELETE /posts/1

GraphQL

RESTful API

http://localhost:3000/posts

<http://localhost:3000/posts>

{ "title": "aa", "content": "bb" }

app.get("/posts")

app.post("/posts")

app.put("/posts")

app.delete("/posts")

서버

GraphQL

요청 구문 { posts

{

title

content

}}

{ "data":

{ "title": "aa",

content: "bbb" }

app("/graphql")

스키마

목록

상세보기

글쓰기

리졸버 삭제

목록반환

상세보기 반환

글쓰기후 결과

반환 서버

GraphQL

1. GET <http://localhost:3000/posts>

id, title, content, author, createdAt

2. GET <http://localhost:3000/v2/posts>

id, title, content, author, createdAt, count

```
posts {  
  title  
  content  
  author  
}
```

배치 요청

GET <http://localhost:3000/posts/2>

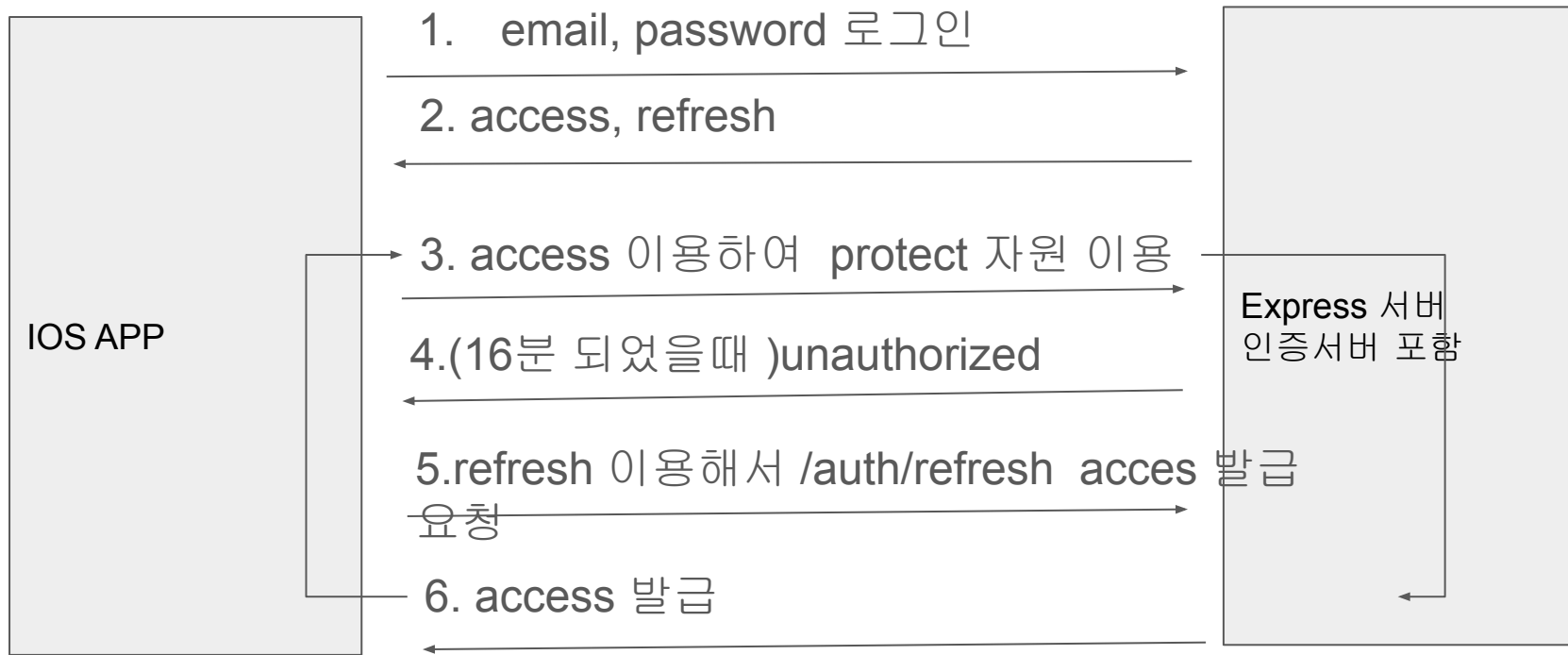
GET <http://localhost:3000/posts/2/comments>

```
{ getPosts(ID:2) {  
  title  
  content  
  auth}  
  getcomments(ID: 2) {  
    content  
  }  
}
```

스키마 정의

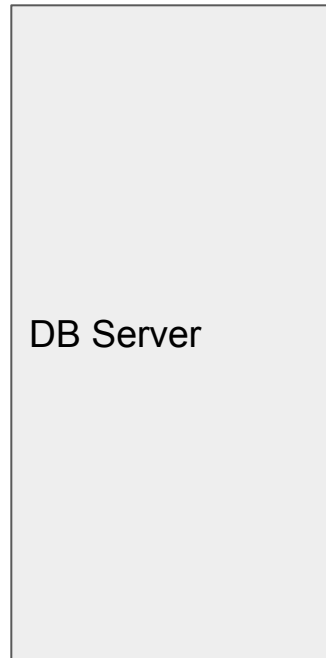
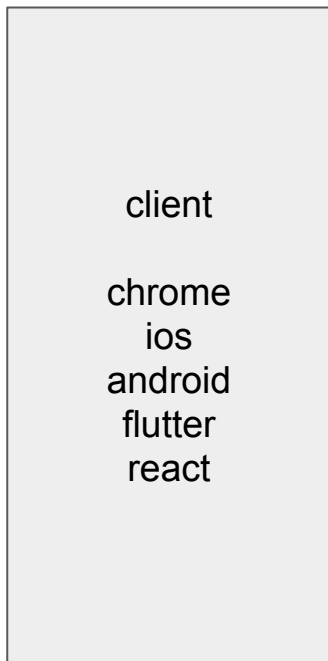
```
type Query {  
  hello: String  
  getPosts(id: ID) : Post  
}  
  
type Post {  
  id: ID  
  title: String  
  content: String  
}
```

access token, refresh token 인증

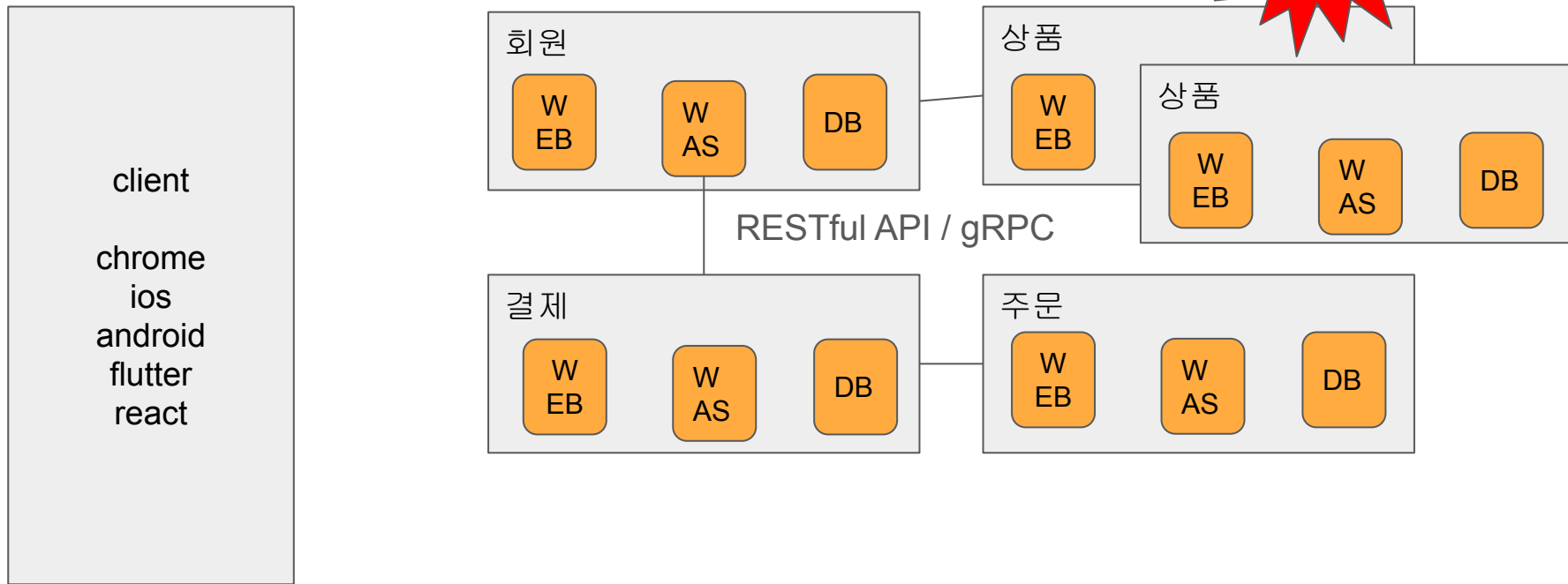


만약 이 작업이 귀찮으면 access token을 '14d' 정도

아키텍처 패턴 - 모놀리식



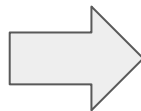
아키텍처 패턴 - 모놀리틱



group by

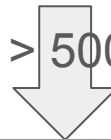
| cust_id B | company | city | mileage |
|--------------|---------|------|---------|
| 1 | A 상사 | 서울 | 100 |
| 2 | B 상사 | 서울 | 200 |
| 3 | C 상사 | 부산 | 300 |
| 4 | D 상사 | 부산 | 400 |
| 5 | E 상사 | 대전 | 200 |
| 6 | F 상사 | 대전 | 500 |

group by



| city | mileage |
|------|---------|
| 서울 | 300 |
| 부산 | 700 |
| 대전 | 700 |

having sum(mileage) > 500



| city | mileage |
|------|---------|
| 서울 | 300 |
| 부산 | 700 |