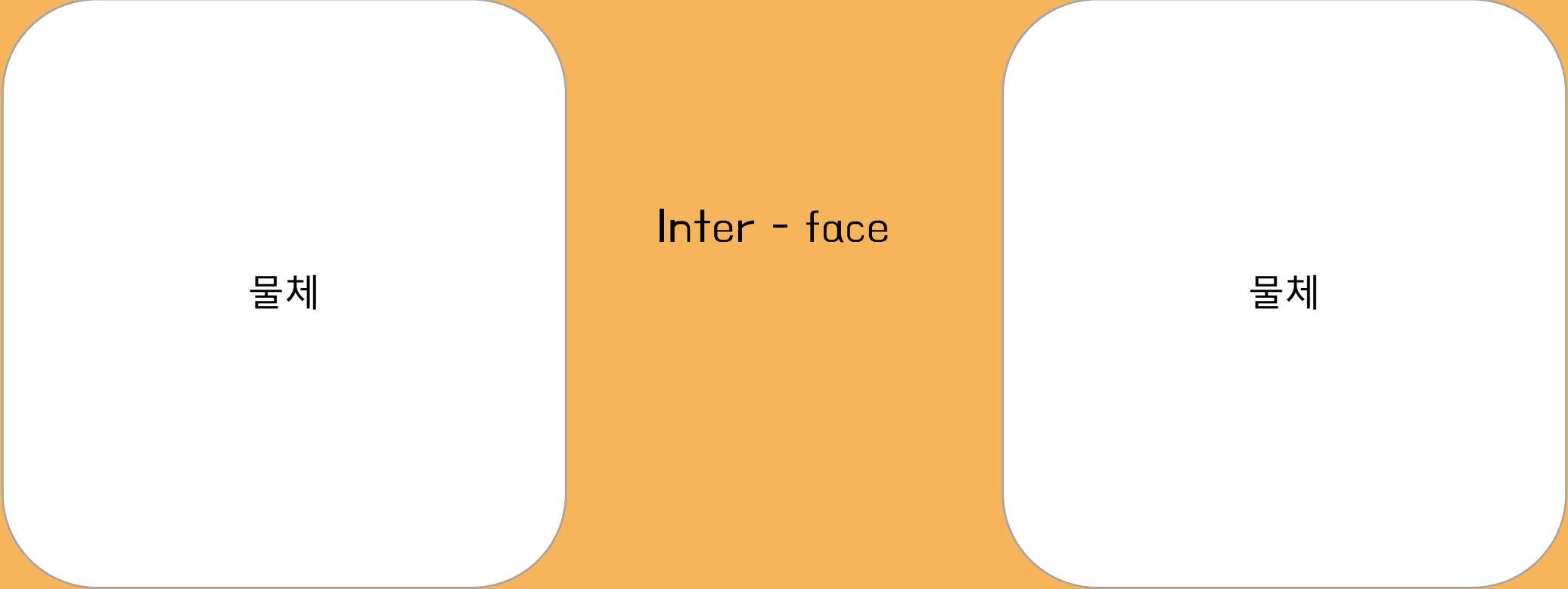


DEVKOR

Backend Study

API

Application Programming Interface



물체

Inter - face

물체

API

Application Programming Interface



program

The diagram consists of two white rounded rectangles, one on the left and one on the right, each containing the word 'program'. Between them is the text 'Inter - face'. The entire diagram is set against a solid orange background.

Inter - face

program

REST API

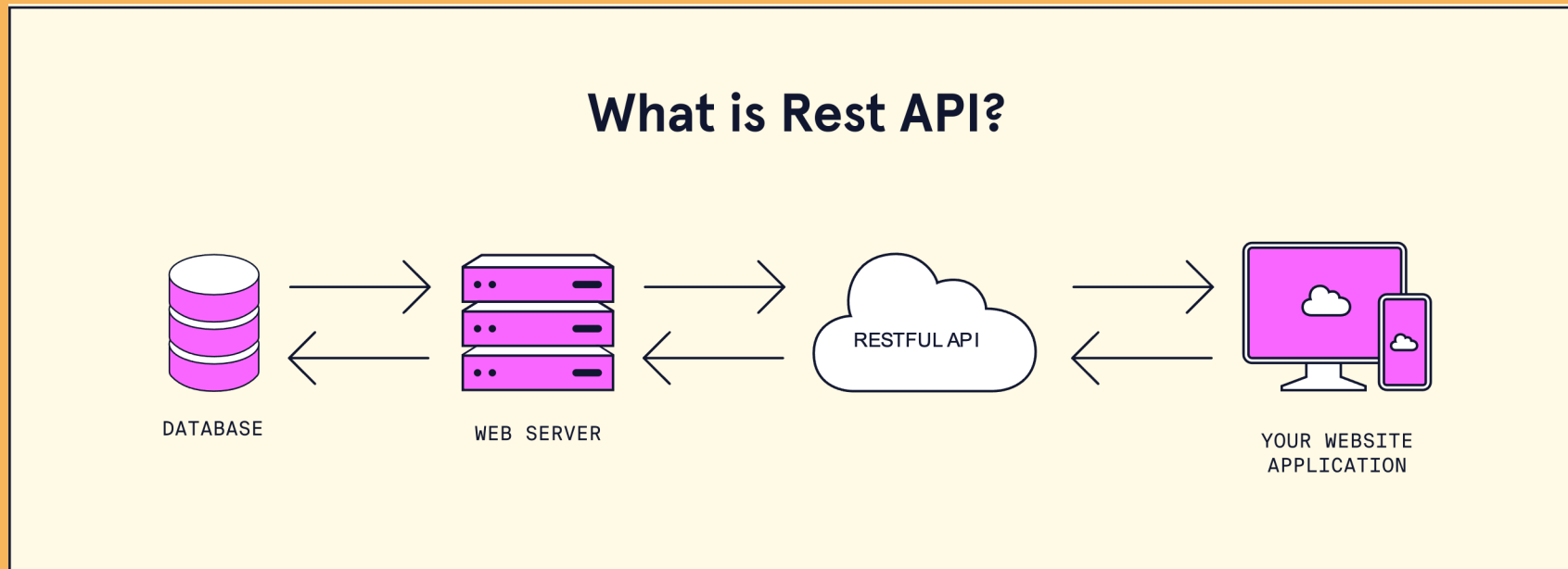
REpresentational State Transfer(-ful) API

로이 필딩이 박사논문에서 소개한 소프트웨어 API 아키텍처

Network 통신에 최적화되어있으므로

그 가장 큰 응용인 웹 기술에서 광범위하게 쓰임,

API 중 REST 조건을 만족하는 API를 말함



REST API의 조건

REpresentational State Transfer
표현, 상태 전송..

- client-server
- stateless
- cache
- uniform interface**
- layered system
- code-on-demand (optional)

REST API의 조건

- client- server

자원을 주고 받는 API

-> 자원이 있는 server, 자원을 요청하는 server의 존재.

server가 API를 제공하고, 로직을 처리.

client가 로그인 정보등의 context를 관리..

클라이언트와 서버로 분리함으로써 서로의 의존성을 줄임

web-> 웹 서버와 브라우저. 당연히 지켜짐

<https://blog.npcode.com/2017/04/03/rest의-representation이란-무엇인가/>

<https://gmlwjd9405.github.io/2018/09/21/rest-and-restful.html>

REST API의 조건

- Stateless

state. 상태 정보를 서버가 저장하지 않음

상태를 저장하지 않는다는 것..

서버는 client가 누군지 모른다 .

- > '요청'만을 보고 어떤 답을 할지 판단함

- > 요청만 같다면,, 같은 응답을 하는 것 (DB 상태가 다르면 다를 수 있음)

- > 서버가 일관적으로 부담없이 작동할 수 있음

- > 서비스의 자유도가 높아짐.

각각의 요청은 독립적임

이전의 메시지 정보 없이도, 어떤 정보인지 알 수가 있다.

HTTP는 stateless protocol이므로. 당연히 지켜짐

<https://blog.npcode.com/2017/04/03/rest의-representation이란-무엇인가/>

<https://gmlwjd9405.github.io/2018/09/21/rest-and-restful.html>

REST API의 조건

- cacheable

cache = 리소스를 여러 번 요청 할 때, 효율성을 높여주는 것

서버가 요청에 효율적으로 응답하기 위해 캐싱이 가능해야 함

HTTP는 기본적으로 캐싱을 지원

캐싱은 주어진 리소스의 복사본을 저장하고 있다가 요청 시에 그것을 제공하는 기술입니다. 웹 캐시가 자신의 저장소 내에 요청된 리소스를 가지고 있다면, 요청을 가로채 원래의 서버로부터 리소스를 다시 다운로드하는 대신 리소스의 복사본을 반환합니다. 이것은 다음과 같은 몇 가지 목표를 달성하게 해줍니다: 모든 클라이언트를 서비스할 필요가 없어지므로 서버의 부하를 완화하고, (캐시가 원래 서버에 비해서) 클라이언트에 더 가까이 있으므로 성능이 향상됩니다. 즉, 리소스를 회신하는데 더 적은 시간이 들게 됩니다. 웹 사이트에서 캐싱은 높은 성능을 달성하는 데에 주요한 요소입니다. 반면에 모든 리소스가 영원히 변하지 않는 것은 아니므로 리소스가 변하기 전까지만 캐싱하고 변한 이후에는 더이상 캐싱하지 않는 것이 중요합니다.

<https://blog.npcode.com/2017/04/03/rest의-representation이란-무엇인가/>

<https://gmlwjd9405.github.io/2018/09/21/rest-and-restful.html>

REST API의 조건

- Layered System

요청에 응답하기 위해 서버는 다중 계층으로 구성될 수 있음

미들 웨어를 사용하듯이..

로드 밸런싱 , 보안, 암호화, 사용자 인증 등 계층을 추가하여, 여러 기능을 할 수 있음

<https://blog.npcode.com/2017/04/03/rest의-representation이란-무엇인가/>

<https://gmlwjd9405.github.io/2018/09/21/rest-and-restful.html>

REST API의 조건

- **Uniform Interface**

uniform한 interface.

리소스가 통일된 기준으로 전달되어야 함 .

- resources requested are identifiable and separate from the representations
 - 리소스는 url로 구분가능해야 함
- resources can be manipulated by the client via the representation
 - 자원이 representation – method + url 에 의해 조작가능해야 함.
- **self-descriptive messages returned to the client have enough information to describe how the client should process it.**
 - **self-descriptive** – 요청 자체로 충분히 정보를 전달해야함
- **hypertext/hypermedia is available**
 - **application의 상태는 hyperlink를 통해 전이되어야 함.**

<https://www.redhat.com/en/topics/api/what-is-a-rest-api>

<https://blog.npcode.com/2017/04/03/rest의-representation이란-무엇인가/>

<https://gmlwjd9405.github.io/2018/09/21/rest-and-restful.html>

REST API의 조건

- Self- Descriptive messages

메세지만 보고, 그 뜻을 알아야 함.

서버가 업데이트 되어 메세지가 변하더라도, 클라이언트는 그 메세지만 보고도 해석이 가능하게 함으로써, 확장 가능성을 완수하자..

Self-descriptive message

```
GET / HTTP/1.1
```

→ 이런 호출은 Self-descriptive message가 아니다. 왜냐하면 목적지가 없기때문

```
Get / HTTP/1.1  
Host: www.example.org
```

→ 이렇게 변경하면 Self-descriptive message가 된다.

```
HTTP/1.1 200 OK  
\[{ "op" : "remove", "path" : "a/b/c"}\]
```

→ 서버가 위와 같이 응답한다면 아래와 같이 바꾸어야 Self-descriptive message라고 할만하다.

```
HTTP/1.1 200 OK  
Content-Type: application/json-patch+json  
\[{ "op" : "remove", "path" : "a/b/c"}\]
```

→ json-patch의 명세를 보면 확인할 수 있다.

즉, Self-Descriptive message(자기 서술적 메시지)란 메시지만으로 어떤 메시지 인지 알 수 있어야 한다.

<https://www.redhat.com/en/topics/api/what-is-a-rest-api>

<https://blog.npcode.com/2017/04/03/rest의-representation이란-무엇인가/>

<https://gmlwjd9405.github.io/2018/09/21/rest-and-restful.html>

REST API의 조건

- HATEOAS Hypermedia As The Engine Of Application State

하이퍼 링크를 이용한
상태 전이

링크를 이용하므로,,
서버가 제공하는 uri가 바뀌어도
클라이언트는 수정할 필요가 없음

```
1 {  
2   "account_id" : 12345,  
3   "balance" : "350,000"  
4 }
```

기존의 전형적인 REST API 응답

```
1 {  
2   "account_id" : 12345,  
3   "balance" : "350,000",  
4   "links" : [  
5     {  
6       "rel" : "self",  
7       "href" : "http://localhost:8080/accounts/12345"  
8     }, {  
9       "rel" : "withdraw",  
10      "href" : "http://localhost:8080/accounts/12345/withdraw"  
11     }, {  
12      "rel" : "transfer",  
13      "href" : "http://localhost:8080/accounts/12345/transfer"  
14     }  
15   ]  
16 }
```

<https://www.redhat.com/en/topics/api/what-is-a-rest-api>

<https://blog.npcode.com/2017/04/03/rest의-representation이란-무엇인가/>

<https://gmlwjd9405.github.io/2018/09/21/rest-and-restful.html>

REST API

- 조건들이 번거로울 수 있다.
- Just. 설계 지침..
- 꼭 지킬 필요는 없다는 것
- 설계자의 의도대로 ~

<https://www.redhat.com/en/topics/api/what-is-a-rest-api>

<https://blog.npcode.com/2017/04/03/rest의-representation이란-무엇인가/>

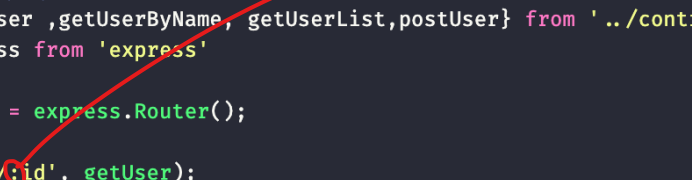
<https://gmlwjd9405.github.io/2018/09/21/rest-and-restful.html>

HTTP 메서드 요청 정보

- GET
 - query- 조건 만족 리소스, params- 특정 리소스 지정
 - -> ?<>=<> & ? <>=<>
 - -> url에 들어가는 정보
- POST
 - body

REST API 설계하기

```
1 import {getUser,getUserByName,getUserList,postUser} from '../controller/user'
2 import express from 'express'
3
4 const router = express.Router();
5
6 router.get('/:id', getUser);
7 router.get('/info', getUserByName);
8 router.get('/', getUserList);
9 router.post('/register', postUser);
10
11 export default router;
```



```
1 export const getUser = (req, res, next) => {
2   const {id} = req.params;
3   // GET user from DB WHERE user.id=id
4   res.json(user);
5 }
6 export const getUserByName = (req, res, next) => {
7   const {name} = req.query;
8   // GET user from DB WHERE user.name=name
9   res.json(user);
10 }
11
12 export const getUserList = (req, res, next) => {
13
14   // GET users from DB
15   res.json(userList);
16 }
17
18 export const postUser = (req, res, next) => {
19   const {user} = req.body;
20   // INSERT user INTO DB
21   res.status(200).json({success:true});
22 }
```

REST API 예시

간단한 리뷰 플랫폼을 위한 API

<https://www.notion.so/overthestream/2022-2-DevKor-c838f012699d4e81975dad0deeeb4ec5>

REST API 과제

\$ git branch -m develop

develop 브랜치를 만들어서
REST API 설계 후, db 로직 제외하고 예시처럼 코딩해 오기
++ -> develop -> main pr 만들어서, 링크 제출!
기간: 이번 주 주말까지

간단하게 아무 주제나 괜찮습니다.
ex) 친구와 채팅 앱, todo list 앱, 일기장 앱 ..

모르는 건 자유롭게 질문해주세요!
끝난 후, pr에 피드백드리겠습니다

감사합니다!