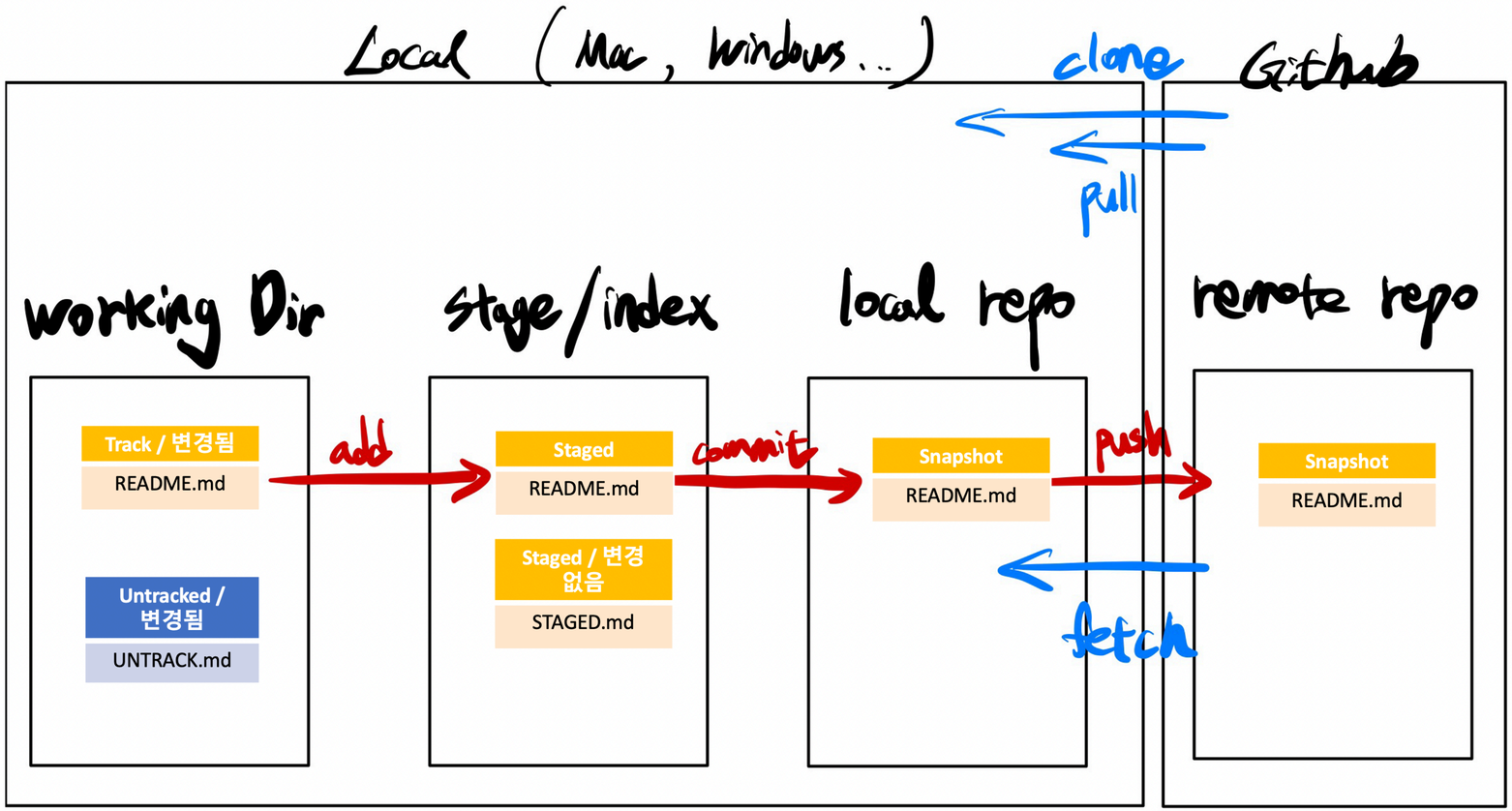
Backend-NestJs 01 - 2

Git 으로 협업하기, http와 rest api의 이해, nestjs 속의 디자인 패턴

컴퓨터학과 홍석민
https://github.com/honghyeong

Git : 기본 원리


Local (Mac , windows . .)
clone
Github
pull
working Dir
Stage/index
local repo
remote repo
Track / 변경됨
README.md
add
Staged
README.md
commit
Snapshot
README.md
push
Snapshot
README.md
Untracked / 변경됨
UNTRACK.md
Staged / 변경 없음
STAGED.md
fetch

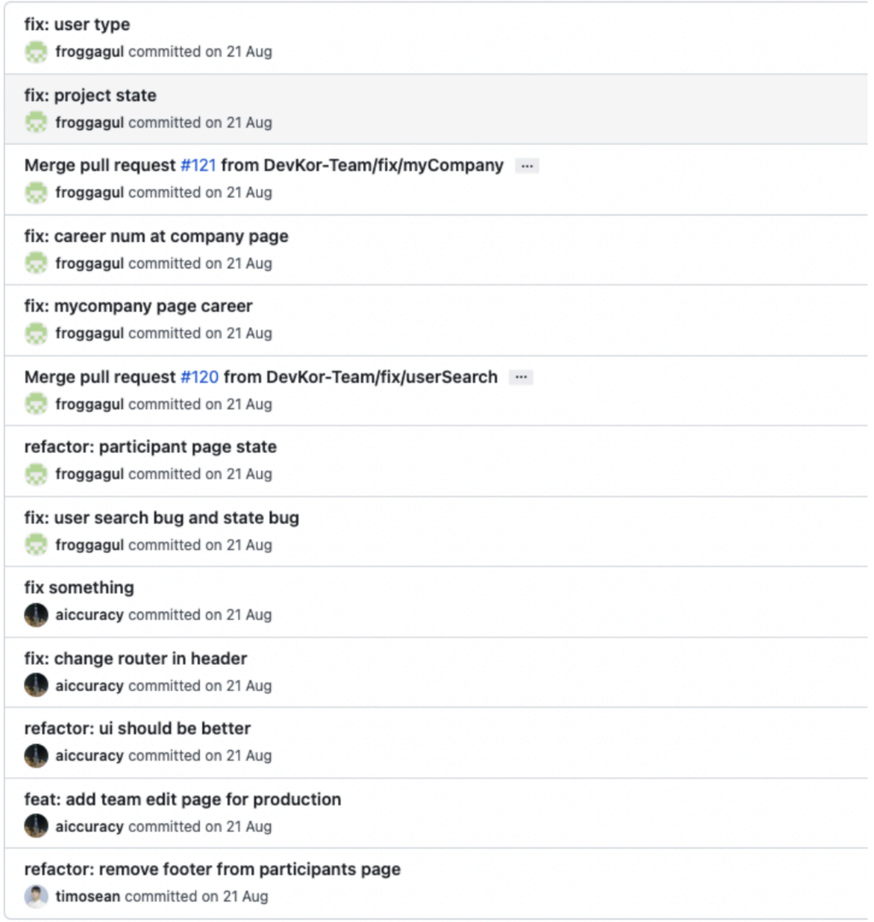
Git : PR, branch

- branch는 왜 파는걸까요 ?
- Branch는 어떻게 파고, 주의할 점은 무엇일까요?
- PR 이 뭘까요 ?
- 왜 Pull Request 라는 이름을 갖고 있을까요?

Git : commit convention

Commit Type

- feat : 새로운 기능 추가
- fix : 버그 수정
- docs : 문서 수정 (Ex. README.md 수정)
- style : 코드 포맷팅, 세미콜론 누락 (Ex. ESLint, Prettier 적용 등)
- refactor : 코드 리팩토링
- test : 테스트 코드 추가
- chore : 빌드 업무, 패키지 매니저 수정, ignore 파일 수정



Devkor 동아리원님들의 훌륭한 commit 메세지

Git : branch naming convention

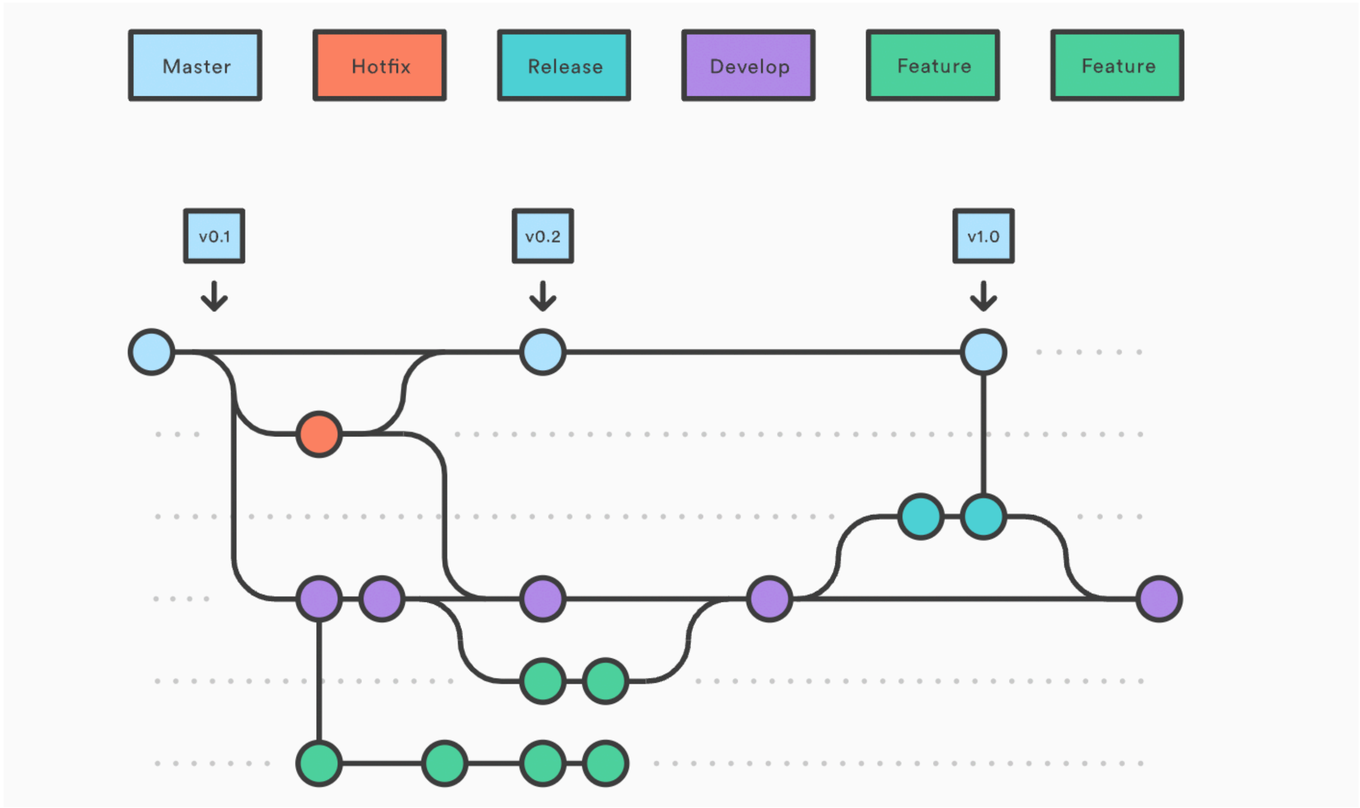
- PR 을 보고 어떤 작업을 하는 브랜치인지 한 눈에 파악
- Merge 된 branch를 보고 어떤 작업이 있었는지 한 눈에 파악

Branch Name

이때 브랜치명은 해당 개발자가 어떤 작업을 하는 브랜치인지 그 큰 카테고리라고 보면 된다.
브랜치 네이밍 컨벤션을 통해서 개발을 하면서 어떤 작업이 있을지 간단하게 살펴보자.

- main(master) : 실제 프로덕션으로 출시된 코드가 존재하는 브랜치
- develop : main 브랜치에 추가하고자하는 새로운 서비스 및 기능을 개발하는 브랜치
- feature : 기능 개발을 하는 브랜치
- hotfix : 치명적인 버그를 빠르게 수정하기 위한 브랜치
- refactor : 리팩토링을 위한 브랜치
- infra : CI/CD, Iac, Docker 등 인프라와 관련된 설정을 하는 브랜치

Git : workflow



Git : Reset, Revert, Cherry Pick

- Reset
  - Soft
  - Mixed
  - Hard

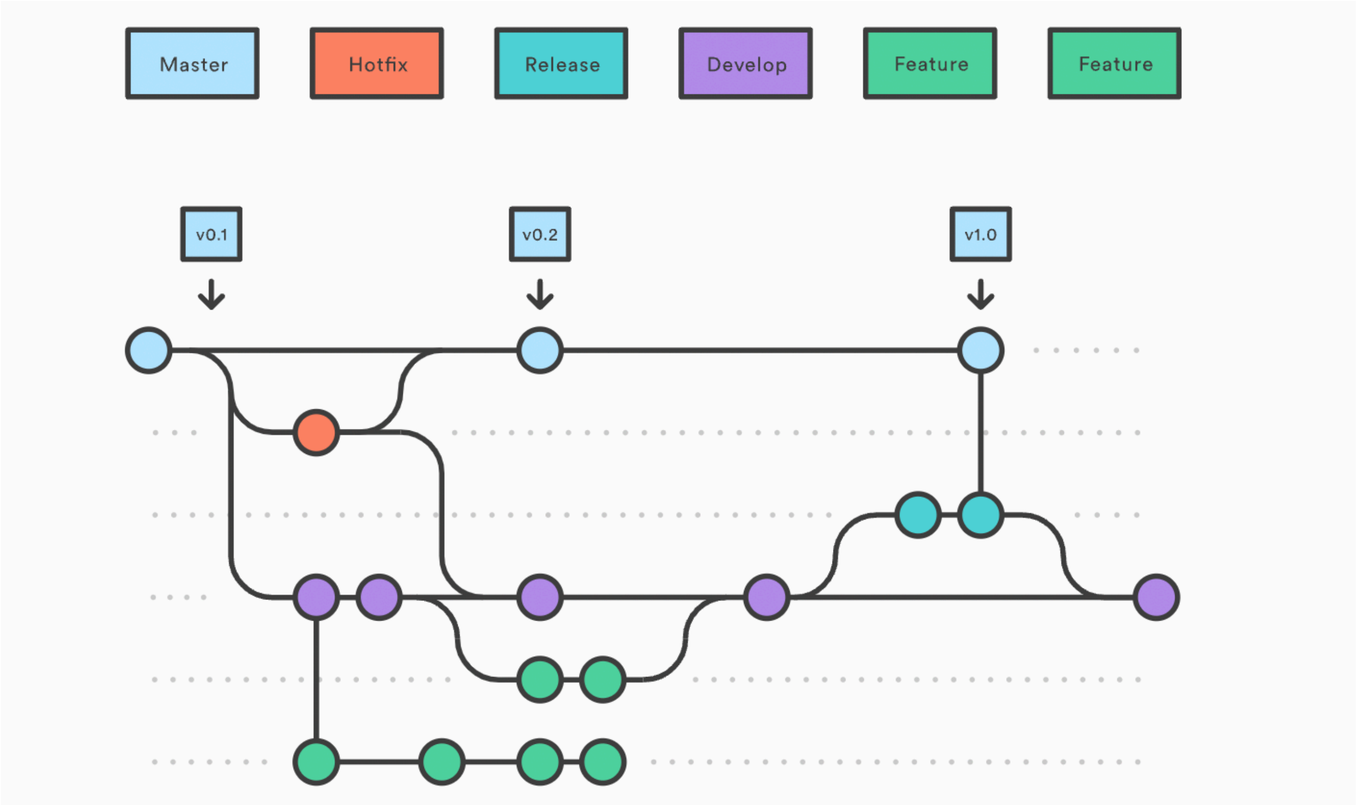
- Forced Push

- Revert

- Amend
  - Amend -m
  - Amend —no-edit

Git : Backmerge

- Backmerge 란?



HTTP

- Protocol : 다른 장치들 끼리 통신을 위해 상호간 맺는 규약
  - 이로써, 각 장치가 어떻게 설계 되었는지 고려할 필요 없이 HTTP 라는 인터페이스에 맞게 설계하면 같은 프로토콜을 기반으로 개발된 장치는 서로 통신할 수 있다.

- HTTP :
  - 전송 계층 위에 있는 애플리케이션 계층
  - World Wide Web 을 기반으로 하는 프로토콜

- Request

- Response

HTTP : Request

- Method
- Protocol
- Host
- Path
- Query
- Headers
- Body

```
GET /search?q=test HTTP/2
Host: www.bing.com
User-Agent: curl/7.54.0
Accept: */*
```

HTTP : Response

- Protocol
- Status Code
- Headers
- Body

```
HTTP/1.1 200 OK
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Transfer-Encoding: chunked
Vary: Accept-Encoding


<!DOCTYPE html>
...
```

API

- API 란?

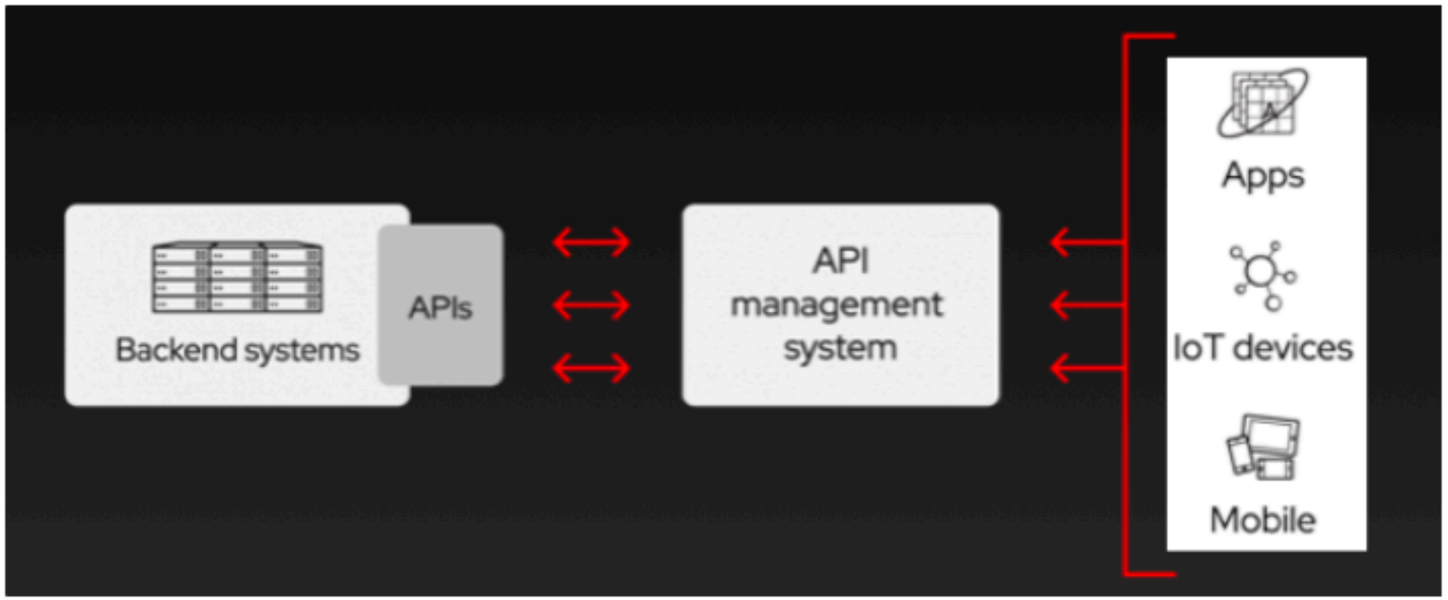
```c
#include<stdio.h>

int main(void) {
  printf("Hello, world!");
}
```

식당 점원은 손님에게 가능한 요리의 목록을 제공하고, 손님의 요청을 요리사에게 전달합니다.

API는 클라이언트 프로그램에 가능한 메소드 목록을 제공하고, 클라이언트 요청을 서버 프로그램에 전달합니다.

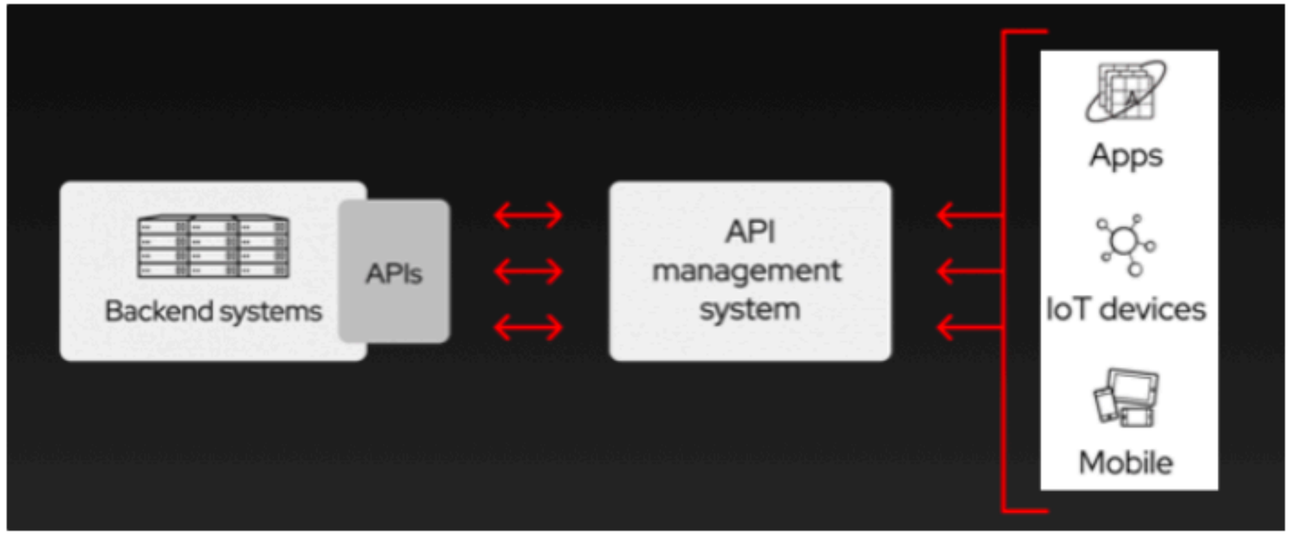
API

- Backend에서 제공하는 API



RESTful ?

- REST (REpresentational State Transfer)
- 백엔드에서 API를 설계할 때 지키는 일종의 약속 중 하나
- 웹의 장점을 최대한 활용할 수 있는 아키텍처로써 REST를 발표했다고 합니다.



RESTful ?

- REST (REpresentational State Transfer) 의 특징
  - Uniform
  - Stateless (Session X)
  - Cacheable
  - Self-descriptiveness
  - Client - Server
  - 계층형 구조

RESTful : Self-descriptive, Uniform

- URI : 정보의 자원을 표현하는 데에 집중
- HTTP Method : 자원에 대한 행위만을 표현

규칙	나쁨	좋음
마지막이 / 로 끝나서는 안 된다.	http://api.test.com/users/	http://api.test.com/users
_ 대신 - 를 사용한다.	http://api.test.com/tag/rest_api	http://api.test.com/tag/rest-api
소문자로 구성한다.	http://api.test.com/tag/REST-API	http://api.test.com/tag/rest-api
동사(Verb)는 포함하지 않고, HTTP Method 로 대체한다.	POST http://api.test.com/delete-user/1 POST http://api.test.com/delete/user/1	DELETE http://api.test.com/user/1
파일 확장자 표시하지 않기	http://api.test.com/users/1/profile.png	http://api.test.com/users/1/profile (`Accept` 사용) `Accept: image/png`

RESTful : Self-descriptive, Uniform

- 무엇이 잘못된 걸까요? 수정해봅시다.
  - 회원 조회 : GET /members/show/1
  - 회원 삭제 : GET /members/delete/1
  - 회원 추가 : GET /members/insert/2

METHOD	역할
POST	POST를 통해 해당 URI를 요청하면 리소스를 생성합니다.
GET	GET를 통해 해당 리소스를 조회합니다. 리소스를 조회하고 해당 도큐먼트에 대한 자세한 정보를 가져온다.
PUT	PUT를 통해 해당 리소스를 수정합니다.
DELETE	DELETE를 통해 리소스를 삭제합니다.

RESTful : Self-descriptive, Uniform

- 유저1이 갖고있는 강의를 조회하려면?

GET : / ~

METHOD	역할
POST	POST를 통해 해당 URI를 요청하면 리소스를 생성합니다.
GET	GET를 통해 해당 리소스를 조회합니다. 리소스를 조회하고 해당 도큐먼트에 대한 자세한 정보를 가져온다.
PUT	PUT를 통해 해당 리소스를 수정합니다.
DELETE	DELETE를 통해 리소스를 삭제합니다.

RESTful : Status Code

- 잘 설계된 REST API는 URI만 잘 설계된 것이 아닌 그 리소스에 대한 응답을 잘 내어주는 것까지 포함되어야 합니다.

- 정확한 응답의 상태코드만으로도 많은 정보를 전달할 수가 있기 때문에 응답의 상태코드 값을 명확히 돌려주는 것은 생각보다 중요한 일이 될 수도 있습니다.

가장 많이 사용되는 200, 201, 400, 400, 500
에러가 어떤 것을 의미하는지 찾아봅시다

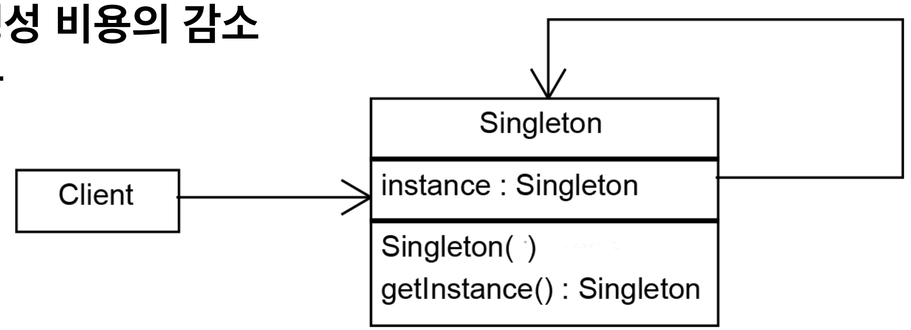
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

RESTful : Status Code

- 마지막으로 **REST API**는 정해진 명확한 표준이 없기 때문에 **REST API**를 사용함에 있어 '무엇이 옳고 그른지'가 아닌 개발하는 서비스의 특징과 개발 집단의 환경과 성향 등이 충분히 고려되어야 한다.

Singleton Pattern

- 하나의 인스턴스를 만들어 놓고 해당 인스턴스를 다른 모듈들이 공유하면서 사용
  - 인스턴스 생성 비용의 감소
  - 의존성 증가



README.md singleton.js

design-pattern > singleton.js > ...

You, 2 days ago | 1 author (You)

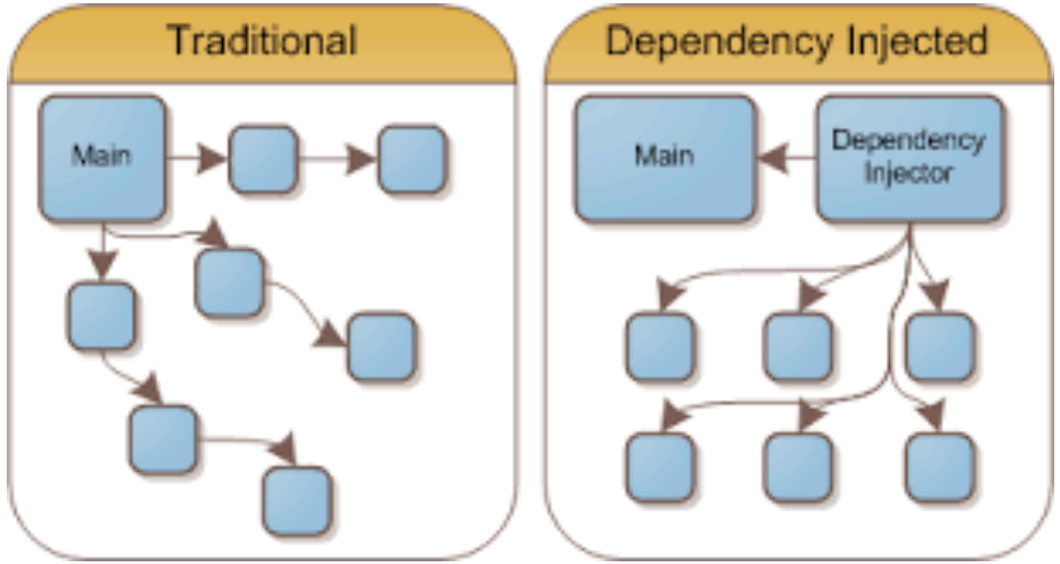
```
const obj1 = {
  a: 27,
};

const obj2 = {
  a: 27,
};

console.log(obj1 === obj2); // false

You, 2 days ago | 1 author (You)
class Singleton {
  constructor() {
    if (!Singleton.instance) {
      Singleton.instance = this;
    }
    return Singleton.instance;
  }

  getInstance() {
    return this.instance;
  }
}

const a = new Singleton();
const b = new Singleton();
console.log(a === b); // true
```

Dependency Injection

- Singleton Pattern 사용하기가 쉽고, 굉장히 실용적이지만 모듈 간의 결합 강하게 만든다는 단점

  - 모듈간의 결합을 조금 더 느슨하게 만들 수 있다.
  - 메인 모듈이 직접 다른 하위 모듈에 의존하는 것이 아니라, 의존성 주입자가 이 부분을 가로채 메인 모듈이 간접적으로 의존성을 주입하도록 한다.

=> Decoupling !
=> Testing, Migration, Module Change



NestJS 설치

https://wikidocs.net/148194

Practice : RESTful API 설계와 git 활용

- Devkor-backend-nest 레포지토리에서 develop branch를 생성하고, default branch로 설정해주세요.
- “01-REST API & Git” 이라는 제목의 Issue를 생성하고, branch를 파서각각의 요구사항에 맞게 REST API 명세를 설계하고, README에 작성해서 PR을 날려주세요

각 사용자는 id(int), name(String), age(int), role(int) 라는 property를 갖고있습니다.

1. 전체 사용자 조회
2. 사용자의 Id를 이용해서 특정 사용자 조회
3. 특정한 id를 가진 사용자의 정보를 body에 전달한 age와 role에 따라서 업데이트
4. 특정한 id를 가진 사용자의 회원 탈퇴

* 단, commit convention과 branch naming convention을 주의해서 PR을 날려주시길 바랍니다.

Practice : RESTful API 설계와 git 활용

METHOD	URI	Params	Body	Description

README 에서 표를 출력하는 것은 각자 구글링을 통해서 알아봅시다. 😇

Reference

REST

- https://meetup.toast.com/posts/92
- https://sharplee7.tistory.com/49
- https://pronist.dev/146

Git

- reset, revert : https://kyounghwan01.github.io/blog/etc/git/git-reset-revert/#revert
- Amend : https://www.atlassian.com/ko/git/tutorials/rewriting-history
- Git workflow : https://lhy.kr/git-workflow