4장 HTTP의 구조 및 핵심 요소

※ 몰랐던 부분, 까먹은 부분 위주로 필기 예정

- HTTP 통신은 Stateless 다.
  - Client 와 Server 가 연결되어 있지 않다.
    - 각각의 통신은 모두 독립적이며 이전 통신은 알지 못한다.
  - 이로 인해 서버 디자인은 간단 해짐 (상태 처리, 저장을 구현 or 관리 하지 않아도 됨)
    - 단점: 기억을 못하니 요청을 처리하기 위해 필요한 모든 데이터를 매번 보내야 됨.
      - ex) 로그인을 기억 못함 ⇒ 사실여부를 Client 단에서 보내야 됨
        - 쿠키, 세션 등
          - 쿠키: Client 쪽 저장 파일
          - 세션: Server 단 저장 파일

- HTTP 요청 구조

1) Start Line
   - 요청의 시작줄, ex) GET /search HTTP/1.1
     - GET: HTTP 메소드 (GET, POST, PUT, DELETE 등)
     - /search: Request Target (요청이 전송되는 목표 주소)
     - HTTP/1.1: HTTP Version

2) Headers
   - HTTP 요청 그 자체에 대한 정보
     - ex) Host 정보, Content-Length: 요청 메세지의 전체 크기, User-agent ...
   - Key : Value 형식 (JSON, 파이썬의 Dictionary 형식)

3) Body
   - 요청이 전송하는 데이터를 담고 있는 부분

- HTTP 응답 구조
  1) Status Line
    ↳ ex) HTTP/1.1 (HTTP Version) 404 (Status code) Not Found (status text)

  2) Headers
    ↳ 요청의 헤더 부분과 동일. (key 값이 서버용인 것들이 있음)

  3) Body
    ↳ 요청의 body 부분과 동일. (보통 웹 콘텐츠 나오는 듯)

- 자주 사용하는 HTTP 메소드
  ↳ GET, POST, OPTIONS, PUT, DELETE
  (PUT를 써서 잘 안씀)

- 자주 사용되는 HTTP Status Code & Text
  ↳ 200 OK, 301 Moved Permanently, 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 500 Internal Server Error

- API 엔드포인트 아키텍처 패턴
  - RESTful HTTP API (Representation State Transfer)
    ↳ 로이 필딩 (Roby Fielding) 박사가 2000년 박사 논문에서 제시한 개념
    - API 에서 전송하는 resource를 uri로 표현, 해당 resource에 행하고자 하는 의도를 HTTP 메소드로 정의하는 방식
    - GraphQL, REST의 over fatching 문제를 하고 만든 게 시원김 P.98참고

Ch5. 본격적으로 API 개발하기

- from flask import request, jsonify
  - ↳ request를 사용해, 사용자가 HTTP 요청을 통해 전송한 JSON 데이터를 읽어들일 수 있다.
  - ↳ jsonify: dictionary 객체를 JSON으로 변환하여 HTTP 응답으로 보낼 수 있게 된다.

- app = Flask(__name__)
  - ↳ import 한 Flask 클래스를 객체화 시켜서 app이라는 변수에 저장 a.k.a. API Application ← 여기에 API 설정, 엔드포인트를 추가하면 API가 완성이다.

- app.users = { }
  - ↳ 새로 가입한 사용자를 저장할 dictionary를 users란 변수에 정의한다.
    key는 사용자 아이디, value는 dictionary에 저장되어 있는 사용자 정보

- app.id_count = 1 (atomic increment operation 필요)
  - ↳ 사용자의 id 값을 저장하는 변수
  - ↳ atomic 연산

- new_user = request.json
  - ↳ HTTP 요청을 통해 전송된 회원 정보를 읽어 들인다.
    request는 엔드포인트에 전송된 HTTP 요청 정보 (헤더, body 등등)을 저장하고 있음
    request.json은 해당 HTTP 요청을 통해 전송된 JSON 데이터를 python dictionary로 변환해준다.

- new_user["id"] = app.id_count
  - ↳ HTTP 요청으로 전송된 회원가입 정보에 id 값을 더하여 준다.

- app.users[app.id_count] = new_user
  ↳ 회원 가입하는 사용자의 정보를 위에 생성한 dictionary에 저장.
  (key는 id, value는 회원 가입 정보)

- app.id_count = app.id_count + 1
  ↳ 다음 사용자가 id+1 이 될 수 있게 한다.

- return jsonify(new_user)
  ↳ 회원 가입한 사용자 정보를 JSON 형태로 전송한다.
  dictionary → JSON 으로 변환

- payload = request.json
  user_id = int(payload['id']) # 해당 사용자의 아이디를 읽는다.
  user_id_to_follow = int(payload['follow']) # 해당 사용자가 팔로우할 사용자의 아이디를 읽는

- user = app.users[user_id]
  # app.users 딕셔너리에서 해당 사용자 아이디를 사용해서 해당 사용자의 데이터를 읽음
  user.setdefault('follow', set()).add(user_id_to_follow)
  # P.114 setdefault 찾아보기
  user가 'follow' 필드를 갖고 있다면, 즉 사용자가 다른 사용자를 팔로우 한적이 있다면, follow 키와 연결되어있는 set에 팔로우하고자 하는 사용자 id를 추가한다.
  ※ set 자료구조는 파이썬의 json 모듈이 JSON으로 변형하지 못함.
  ↳ 커스텀 JSON encoder 를 구현해 사용해야 함. P.116

6장 데이터베이스

- 정규화 normalization : 중복을 최소화해 데이터를 구조화하는 프로세스
- 트랜잭션 transaction : 일련의 작업들을 묶어서 하나의 작업으로 실행하는 것

ACID의 성질을 가짐

RDB

- 데이터를 더 효율적이고 체계적으로 저장하고 관리 가능
- 미리 저장하는 데이터들의 구조를 정의함, 데이터의 완전성 보장
- 트랜잭션 제공
- 데이터를 미리 정의해야 함, 유연성↓
- 서버 확장 보다 성능을 올리는게 효율이 높음.

NoSQL

RDB의 반대.

간단하고 유연함.

구조적이 못하고 트랜잭션을 제공해주지 않는다.

나머지는 SQL 문 설명 및 MySQL 사용 및 API↔DB 연결 작업 설명

P. 145 ~ 171

↳ SQLAlchemy 사용

7장 인증

- 비밀번호를 암호화해서 DB에 저장해야 하는 중요성..!
- import hashlib

  # 모듈 임포트, 단방향 해시 암호화를 실행

  m = hashlib.sha256()

  # sha256 암호 알고리즘을 선택

  m = update (b "test password")

  # update 메소드는 바이트(byte) 값을 받아야 함 b prefix를 스트링 앞에 붙여서 변환

  m.hexdigest()

  # 암호화된 값을 16진수 값으로 읽어 들인다.

bcrypt 암호 알고리즘

↳ 단방향 알고리즘의 여러 단점들 때문에 나온 알고리즘. (해킹위험, 속도 문제등)

↳ salting + key seretching 구현한게 bcrypt.

P. 181 사용법

access token -> 로그인 성공 후 Client가 server에 요청에 같이 보내면서 로그인임을 인증하는 방법을 위한 토큰 -> 로그인 여부 확인

JWT : JSON Web Token, access token의 일종으로, JSON -> token으로 변환

P. 182

- JWT 구조

  header payload , signature

  xxxxx.yyyyy.zzzzz

  - header

    ↳ 토큰 타입(JWT)와 해시 알고리즘(hashing algorithm)을 지정

ex) "alg" : "HS256",
"typ" : "JWT"

· payload

↳ JWT를 통해 실제로 서버간에 전송하고자 하는 데이터부분 = HTTP의 body

ex) "user_id" : 2,
"exp" : 1539517391

↳ Base 64URL 코드화 되어서 JWT의 두번째 부분을 구성. 이는 코드화지 암호화가 아니다. 민감한 정보 X

· signature

↳ JWT가 원본 그대로 라는 것을 확인할 때 사용하는 부분

↳ 역시 마찬가지로 Base64URL로 코드화 되어 있음.

PyJWT

· 파이썬 사용하는 JWT 라이브러리, 사용법 P.39

이후 sign-up API에 인증 엔드포인트 구현
및 login

decorator 함수 만들어 사용등을 한다.

Ch.08 unit test

· 테스트는 자동화가 제일 중요하다...!

↳ 메뉴얼 테스트 다음의 장점이 있다. ex) 직관적으로 계획 없이 실험가능

↳ 단점으로는 실행 속도가 느리고, 부정확하고, 다시 할 확률이 높다.

→ 그렇다면 소규모 기업, 스타트 업들은 자동화가 더 중요!

테스트 종류로는 performance test, load test 등등 있지만 function 테스트로는 크게:

· UI test, End-To-End test → 서버를 구성하는 모든 시스템을 실행하고 연결한 후 테스트 = 복잡, 어려움
→ 자동화도구 Selenium 이 있지만, 100% 자동화는 조금 힘들

· integration test → 여태껏 실습에서 해왔던것, 개발하고자 하는 서버를 실제로 실행 하는 테스트

· unit test → 쉽고, 빠르고; 디버깅이 쉽다! 파악이 쉬우니!; 전체 서버를 확인하기는 어려움

ex) def multiply_by_two(x):
    return x * 2

To test:

ex) assert multiply_by_two(2) == 4

↳ assert 그 다음에 오는 표현이 false 일 경우 AssertionError 'exception'을 던짐.

Pytest

P219 시작

· pytest는 'test_'라고 되어 있는 파일들만 테스트 파일로 인식함.

ex) def multiply_by_two(x):
    returum x * 2

↳ testing def test_multiply_by_two():
    assert multiply_by_two(4) == 7

결과! 이렇게 정상 이동하지 않고 터미널에 'pytest'을 입력하면 테스트 자동 실행

· flask에서 HTTP request unit test를 위한 .test_client() 메소가 있다!

매너 API unit test 실습 P-224 ~ 238

· 테스트 할 때는 본격적으로 작성 해야될 코드들이 있을 수 있음. (사용자 생성, 로그인 등)

↳ 각 테스트는 독립적이어야 하고, 서로 영향을 주면 안되기 때문이다.

ex) tweet 엔드포인트 테스트 코드에서 생성된 사용자를 지우지 않은 다면, 다음에 동일한 사용자를 생성할 때 이메일 주소가 같아 오류가 날 것이다.

↳ 그래서 pytest 에는 각 테스트를 실행하기 전과 후로 자동으로 시작해주는 setup_function(), teardown_function() 이 있다.

↳ setup_function에 사용자를 생성, teardown 에 사용자를 (test 데이터)를 삭제 하면 된다.

Ch. 09 AWS에 배포하기

- 사용할 AWS 서비스들은 EC2 (Elastic Compute Cloud), RDS (RDd Service), ALB (Application Load Balancer)

- RDS 실습 P.247 부터...
- EC2 실습 P.259 부터...
- ALB 실습 P.293 부터....
  - Resource clean up! 까먹지 말고 제발... ㅠㅠ

- 배포를 하려면 production 서버에 적합하게 개발해야 함..
  - 배포 코딩 실습 P.266

- 양이 방대하고 중요한 서비스이니 따로 공부하는게 좋을 듯하다.!
  - production 서버 코딩은 연습해보기로 결정!
    - AWS 서비스 개념과 구조는 이해하고 넘어가기!

Ch.10 API 아키텍처

- 아키텍처: 코드의 구조를 체계적으로 & 효율적으로 구현하는 것
  - 아키텍처 요소들 : 확장성 (extensibility), 재사용성 (reusability), 유지보수 가능성 (maintainability), 가독성 (readability), 테스트 가능성 (testability)

- 레이어드 패턴 (이미 정석화된 문제이고, 다른 패턴들로 만들어진 웹앤드 API 에 제일 많이 쓰이는 패턴이다.)

레이어드 패턴 구조 (일반적으로)

Presentation Layer : 클라이언트 시스템과 직접적으로 연결되는 부분 (엔드포인트, 요청 읽고 조작)
↓
Business Layer : 비지니스 로직 (tweet 이 300자가 넘는지)
↓
Persistence Layer : DB 관련로직

- 레이어드 아키텍처 적용 실습 P.290

- DAO, ORM 등
  - P.296
  - P.154
- unit test 들로 모델 얼굴 만들면 좋음

Ch.11 파일 업로드 엔드포인트

- AWS S3에 저장하는 파일 업로드 엔드포인트 구현 예정
  - ↳ 사진을 HTTP request post 할때는 JSON (application/json content type) 의 post 가 아니라 multipart/form-data 의 content type 을 가진다. Body 부분에서 작업 가능
    - ↳ 'boundary' 를 사용해서 body 를 여러 부분으로 나눌 수 있다. that's why P.335
  - ↳ 좀 어려운 느낌 있는..

Ch.12 백엔드면 이것도 해보자!

- 자료구조 & 알고리즘
- DB
- DB migration - DB schema 형상 관리 ; sqlalchemy ORM, Flayway, Liquibase... like
- micro service architecture - 인프라 스트럭처 아키텍처 P.392
- 리눅스 & 데브옵스