7월 21일 실습 강의 보조 자료

1. Flask 복습

목표 : Flask를 이용하여 User의 CRUD 구현하기

Flask?

- 웹 애플리케이션 개발(백엔드 부분)을 위한 파이썬 프레임워크
  - MVC 아키텍쳐 개발
  - REST API 개발
- Django와 달리 가벼운 프레임워크이기 때문에, 거의 모든 것을 처음부터 세팅해야합니다
  - 그럼 왜 Django 안쓰고 Flask써요?
  - 백엔드 개발자가 되려면 Spring해야한다는데, Flask는 도움이 안되지 않을까요?
  - ⭐ **프레임워크는 중요하지 않습니다**
  - https://github.com/kamranahmedse/developer-roadmap

CRUD?

- Create
- Read
- Update
- Delete

각각에 해당하는 HTTP 메소드에는 무엇이 있을까요?

CRUD 중 UPDATE의 HTTP METHOD

- PUT과 PATCH의 차이점은 무엇일까요?

구현해야 할 것

```
CRUD
- Create : names에 존재하지 않는 username이라면 추가하기
```

```
- Read   : user.html을 화면에 렌더링하고, names를 템플릿에 전달하기
- Update : names에 존재하는 유저의 이름 변경하기
- Delete : names에 존재하는 유저의 이름 삭제하기
```

2. Flask와 MongoDB 연결하기

목표 : MongoDB에 대한 이해와 Flask와 연동하기

MongoDB?

- NoSQL이란 무슨 뜻일까요?
- RDBMS와 다른 특징?

Flask에서 MongoDB 사용하기

```
# create DB or connect DB
db = client.elice
# db = client['elice']
print(client.list_database_names())

# create Collections or connect Collections
memos = db.memos
# memos = db['memos']

# 위 2개가 제대로 생성되었는지 확인해보자. ( 결과가..? )
print(client.list_database_names())
print(db.list_collection_names())


# 데이터 삽입
memos.insert_one(new_memo)

# Collection의 모든 데이터 조회
# 여기서 주의해야 할 점은?
memos.find()
```

3. 크롤링과 Flask

목표 : Flask에서 크롤링 성공하기

- Headers의 UserAgent는 무엇일까요?

    - 크롤링시 봇을 감지하여 차단하는 기능을 막기위해 사용합니다
    - 서버는 User Agent 검사 등의 방법으로 일반 사용자(사람)와 봇을 구분하여 차단할 수 있습니다.
    - 유저 에이전트란?
        - 브라우저가 웹사이트에 연결을 시작할 때 전달되는 기기 정보로 브라우저의 유형, 운영체제 등의 정보가 담겨있습니다.
    - 브라우저의 콘솔창에 `navigator.userAgent`로 확인이 가능합니다.
- 쉽게 데이터를 크롤링하는 방법은 무엇일까요?
    - 페이지 소스보기, 페이지 요소 검사를 이용하여 selector 가져오기
    - BeautifulSoup을 사용하여, html 문서 파싱하기
    - 처음에 가져온 DOM Selector를 사용하여 원하는 데이터를 얻기

4. Hash

목표 : Hash에 대한 이해와 사용 이유 이해. 보안의 중요성 이해.

Hash?

- 해시함수(해시 알고리즘)를 이용하여 고정된 길이의 암호화된 문자열로 바꿔버리는 것
- 해시함수(hash function)는 임의의 길이의 데이터를 고정된 길이의 데이터로 매핑하는 함수

```
ex) 길이가 7인 데이터로 매핑하는 해시함수가 존재한다면?
123             -> abcdefg
2               -> acefghh
45644           -> fgidhsh
44444444444444 -> safagkh
```

- 해시 값을 통한 복호화는 불가능
- Hash를 사용하는 이유?
    - 데이터베이스에 원문으로 비밀번호를 저장하면, 데이터 베이스 관리자가 유저의 비밀번호를 볼 수 있음
    - 해킹과 같은 유출이 일어나면, 원문으로 저장시 비밀번호가 그대로 노출 됨.

- "개인정보의 기술적·관리적 보호조치 기준법" 제 6조 1항에 따르면, "정보통신서비스 제공자등은 비밀번호는 복호화 되지 아니하도록 일방향 암호화하여 저장한다." 라는 법이 존재합니다.

- **원문이 같다면, Hash 값도 같기 때문에 이를 통해 저장 된 비밀번호를 비교**하는데 사용
  - 여기서 발생할 수 있는 문제가 무엇일까요?
    - 레인보우 테이블
    - 무차별 공격 (Brute Forcing)

Flask에서 Hash를 사용하는 방법

```
from werkzeug.security import generate_password_hash, check_password_hash

# generate_password_hash를 통해, 원문을 Hash값으로 변경
# check_password_hash를 통해, 원문과 Hash된 비밀번호를 비교
```

- `generate_password_hash`의 특징
  - 암호화 알고리즘으로 SHA-256을 사용
  - Salt와 Round를 사용
    - Salt는 랜덤으로 생성되는 일정한 길이의 문자열 (default length = 16)
    - 이 Salt 값이 평문 암호에 추가되어 같이 Hash 처리된다.
    - Round는 Hash를 더 안전하게 사용하기 위한 방법
    - Salt와 합쳐진 원문을 다시 Hash하고, 그 값에 다시 Salt를 더해서 Hash하는 것을 반복하는 방법
    - 이 결과로 나온 값은 **'암호화알고리즘:round$salt$해시값'** 이고, 이 데이터를 저장한다.

```
ex)

원문 : abcde

(abcde + salt 값) -> Hash 처리 -> 결과 값 : rsnasfkaskaf

Round 1
(rsnasfkaskaf + salt 값) -> Hash 처리 -> 결과 값 : afgddeghdase
```

```
Round 2
(afgddeghdase + salt 값) -> Hash 처리 -> 결과 값 : kasfksakakaf

.... 이런식으로 N회 반복하며, generate_password_hash는 기본적으로 50000번 반복
```

- check_password_hash의 특징

```
match = check_password_hash(hashed_passsword, plain_password)
```

  - Hash된 값의 Salt와 Round를 통해 원문의 Hash값을 비교하여 True, False를 반환

5. JWT

목표 : JWT를 통한 로그인과 권한 인증에 대해 이해하기

JWT?

- JWT는 일반적으로 클라이언트와 서버, 서비스와 서비스 사이 통신 시 권한 인가(Authorization)를 위해 사용하는 토큰 (문자열)
- 구조 : `HEADER.PAYLOAD.SIGANATURE` 의 구조로 이루어진 문자열
- `Header`
  - JWT를 검증하는데 필요한 정보를 가진 JSON 객체를 Base64 인코딩한 문자열
  - typ : 토큰의 타입을 지정. 당연히 JWT입니다.
  - alg : 해싱 알고리즘을 지정. 보통 SHA-256을 사용합니다. 이 alg는 JWT를 검증할 때 사용하는 Signature에 사용됩니다.

```
{
  "typ":"JWT",
  "alg":"SHA256"
}
```

  위 JSON을 Base64 인코딩한 문자열입니다.

- `Payload`

- JWT의 내용입니다. 이 또한 JSON 객체를 인코딩한 문자열
- iss: 토큰 발급자 (issuer)
- sub: 토큰 제목 (subject)
- aud: 토큰 대상자 (audience)
- exp: 토큰의 만료시간 (expiraton), 언제나 현재 시간보다 이후로 설정되어있어야 합니다.
- nbf: Not Before 를 의미하며, 토큰의 활성 날짜와 비슷한 개념. 이 날짜가 지나기 전까지는 토큰이 처리되지 않습니다.
- iat: 토큰이 발급된 시간 (issued at), 이 값을 사용하여 토큰의 age 가 얼마나 되었는지 판단 할 수 있습니다.
- jti: JWT의 고유 식별자로서, 주로 중복적인 처리를 방지하기 위하여 사용됩니다. 일회용 토큰에 사용하면 유용합니다.

이 외에도 인증에 필요한 데이터를 직접 추가 할 수 있으며, `create_access_token`의 `identity`를 통해 데이터를 추가할 수 있습니다.

보통 `identity`는 유저의 아이디 혹은 권한 (학생, 선생님, 관리자..) 값을 포함합니다.

- `Signature`
  - Base64로 인코딩 된 헤더와 토큰을 Secret 값을 통해 Hash한 값입니다.
  - 이 Signature로 JWT를 검증합니다.

Flask에서 JWT 사용하기

flask_jwt_extended를 사용합니다.

```
from flask_jwt_extended import (
    JWTManager, jwt_required, create_access_token, get_jwt_identity)
```

- `JWTManager`
  - app에서 JWT를 사용할 수 있게 등록하는 함수입니다.
- `@jwt_required()`
  - 라우터에 사용하는 어노테이션입니다.

    - 요청시 JWT가 필요한 (권한 인증이 필요한) 라우터에 사용합니다.
    - JWT를 포함하지 않고, jwt_required가 붙은 라우터에 요청을하면 요청을 거부합니다.
- `create_access_token`
    - Access Token이라는 이름의 JWT를 생성합니다.
    - 생성시 identity와 만료일자를 지정할 수 있습니다.
    - Access Token은 jwt_required가 붙은 라우터에 요청을 할 때 헤더에 아래와 같은 구조로 함께 보내져야합니다.

```
Authorization : Bearer {access token}
```

- `get_jwt_identity()`
    - `@jwt_required()`가 붙은 라우터에서 요청으로 받은 JWT의 identity 값을 반환합니다.

6. ORM과 로그인

목표 : 여태까지 학습한 RDB, ORM, Hash를 통해 로그인 방법 이해하기

데이터베이스

- 데이터 베이스 모델을 통해, 테이블을 구성하고 생성하는 법을 알아보겠습니다.
- ORM을 이용하여 데이터를 저장하고, 가져오는법을 알아보겠습니다.

세션, 로그인, 로그아웃

- Flask Session 사용
    - 세션을 통해 로그인이 유지 되도록 해봅시다.
- Jinja에 Session 적용방법
    - session 변수를 어떻게 Jinja에서 가져오는지 알아봅시다.
    - 세션을 통해, Jinja를 제어하는법을 알아보겠습니다.
- Remember Me 적용방법

    - `session.permanent` 를 통해 브라우저가 종료되어도 세션이 유지되거나 삭제되도록 해봅시다.
- 로그아웃
    - 로그아웃시 세션을 삭제하는법을 알아보겠습니다.

프로필

- 세션이 있으면, 프로필을 통해 user의 데이터를 보여주는 방법을 알아보겠습니다.