# 협업 심화 활동

## 3 주차

### 배민근 백지훈 임경섭 정회훈

# 2018-11-09 회의

백엔드 / 프론트엔드로 나누어 개발을 하려고 하니
기존에 기능 별로 역할을 나눈 방식이 모호함

그래서

서버단 구현은 기능하나씩 다같이 개발하고
프론트 구현은 앞서 나눈 것 처럼 나누어서 구현 하기로 함

# 2018-11-09 회의

서버단 기능 구현 순서

1. 유저 가입 / 인증 / 세션 처리
2. 게시판 기능
    i. 글/댓글 기능
    ii. 글 파일 업로드
3. 게시판 속성 부여, 접근제어

프론트

- 회원가입/로그인 회훈선배
- 게시판 경섭선배
- 어드민 백지훈
- 디자인 배민근

## 유저 데이터 정의

로그인, 케이웹 회원 인증, 탈퇴
기본정보 (이름, 닉네임), 케이웹 상태 (준/정/휴/명/탈, 기수)

우선은 가입하자마자 인증 ok 상태로 저장
이후 관리자 승인 or @korea.ac.kr 이메일 인증으로 인증 구현

# 유저 데이터 정의

- login id
- login credential
- login status
- unverified -> 인증 X
- active -> 인증 ok
- paused -> 휴회원
- ~~terminated -> 탈퇴~~ -> 탈퇴 상태는 별도로 두지 않음.
  다만 탈퇴시 작성한 글은 소유자가 없는 상태로 계속 유지
- email
- kweb
  - status (준/정/명)
  - generation (기수)

## 유저 테이블 정의

다중 로그인, 별칭 이메일, 다중 신분 등의 기능이 없으므로

위 데이터를 모두 하나의 테이블로 관리해도 무방

## 유저 테이블 정의

유저의 비밀번호는 해쉬 알고리즘을 통해 digest 한 hex string을 저장하고 이를 인증에 사용한다.

직접 구현하기 보다는 이미 검증되고 많이 사용되는 구현체인 scrypt를 사용하기로 한다.

유저 credential은 plain text password를 scrypt 라이브러리에 { N: 17, r: 8, p: 1 } 파라메터로 암호화후 버퍼를 hex 스트링으로 저장 (출력길이 192byte)

참고 : 안전한 패스워드 저장

# 유저 테이블 정의

유저 계정 상태, 케이웹 회원 상태 정의

```
enum UserStatus {
    UNVERIFIED, ACTIVE, PAUSED
}
```

```
enum UserKwebStatus {
    ASSOCIATE, REGULAR, HONORARY
}
```

# 유저 테이블 정의

```
CREATE TABLE User (
    id INT PRIMARY KEY AUTO_INCREMENT,
    identity VARCHAR(25) NOT NULL,
    credential VARCHAR(192) NOT NULL,
    status TINYINT NOT NULL DEFAULT 0,
    email VARCHAR(255) NOT NULL,
    kweb_status TINYINT NOT NULL DEFAULT 0,
    kweb_gereration INT NOT NULL DEFAULT 1
);
```

## 유저 인증 방식

유저 인증 방식에는 Cookie Session 과 JWT 두가지 방식이 있다.

두가지 인증 방식 모두 널리 쓰이는 방법이다.

- Cookie : 서버가 세션키를 발급하여 쿠기에 저장. 서버는 쿠키의 세션키를 사용하여 클라이언트를 판별
- JWT(Json Web Token) : 서버는 인증 정보가 담긴 토큰을 발급하며, 헤더를 통해서 들어오는 토큰으로 클라이언트를 반별

# 유저 인증 방식 - 쿠키

Cookie 세션은 세션의 유지, 관리를 위해서 서버가 모든 세션 정보를 기억하고 있어야 한다.

이를 메모리나 데이터 베이스에 저장하게 되는데 메모리에 저장하게 되면 서버가 재시작되거나 메모리가 부족한 경우 세션을 유지 할수 없게 된다.

데이터 베이스에 저장하는 경우에는 서버장애로 인한 세션유실에서 비교적 자유로우나 매번 접속하는 요청마다 데이터베이스를 조회해야 하기 때문에 오버헤드가 발생한다.

**두가지 문제는 세션을 데이터 베이스에 저장하고 자주 확인되고 있는 세션 정보를 메모리에 캐싱 함으로써 보완할수 있다.**

# 유저 인증 방식 - JWT

JWT는 Json Web Token의 약자로 유저가 인증을 요청하면 서버가 인증된 사실에 대한 내용과 해당 내용이 서버에의해 검증 되었음을 나타내는 signature를 포함한 token을 발급한다. 이후 유저는 서버로의 모든 요청에 헤더로 토큰을 삽입하고 서버는 내용과 signiture 만을 확인 함으로써 유저를 확인한다.

**JWT의 단점은 token 발급과 동시에 토큰의 수명이 정해지기 때문에 수명이 끝난 토큰을 갱신해주어야 하고 수명을 너무 길게 설정 할 경우 보안 이슈가 발생한다는 단점이 있다.**

## 유저 인증 방식

오픈소스로 유지될 본 프로젝트의 특성상 JWT 토큰의 서버 사이드 secret을 잘 유지하기 어렵고 프론트엔드 단의 구현이 늘어나는 관계로 유저 인증 방식은 Cookie-session 방식을 택한다.

구현에 있어서는 먼저 인메모리 방식의 세션 저장 방식을 사용하고
추후에 데이터베이스, 캐싱을 구현

## Database Driver, Access

DBMS는 mysql 계열로 결정 되었으니 드라이버는 mysql 패키지를 사용한다.

Data의 Object Mapping을 위하여 ORM 라이브러리를 사용할수도 있으나, Node.js를 통한 데이터 베이스 엑세스를 경험해 보기 위해 직접 DAO를 작성해 보기로 한다.

단, 추후 확장성을 위해서 Repository와 Model을 미리 정의해둔다.

## User 관련 API

유저 관련 API 엔드포인트는 우선 관리자 영역을 베제하고 유저 입장에서만 구현

# User 관련 API - 회원가입

POST

/api/user/join

```
interface UserJoinReq {
    identiy: string;
    credential: string;
    email: string;
}
```

# User 관련 API - 로그인 / 로그아웃

POST - 로그인

/api/user/login

```
interface UserLoginReq {
    identiy: string;
    credential: string;
}
```

GET - 로그아웃

/api/user/logout

## User 관련 API - 비빌번호 변경

PUT

/api/user/credential

```
interface UserCredentialUpdateReq {
    credential: string;
    newCredential: string;
}
```

## User 관련 API - 정보 업데이트

PATCH

/api/user/info

```
interface UserInfoUpdateReq {
    email: string;
}
```

## User 관련 API - 회원탈퇴

POST

/api/user/terminate

```
interface UserTerminationReq {
    credential: string;
}
```

# 깃헙 프로젝트

협업 스터디 레포

벡엔드 서버 레포

감사합니다