2024.04.10

mariadb 코드 작성

```
const mariadb = require('mysql2');

const connection = mariadb.createConnection({
    host : 'localhost',
    user : 'root',
    password : 'thdud852^^',
    database : 'Bookshop',
    dataString : true
});

module.exports = connection;
```

STATUS CODE를 하드코딩으로 작성할 경우, 에러가 발생할 수 있음. 그래서 npm에서 http-status-codes를 설치함.

```
npm i http-status-codes
```

node.js 패키지 구조

app.js : 프로젝트의 메인 라우터 역할

/routes
/users.js : 하위 라우터 역할

/books.js : 하위 라우터 역할

...

경로를 찾은 다음 역할 = 콜백함수를 빼내자!

라우터가 로직까지 다 ~ 수행할 때 단점
1) 프로젝트 규모가 커질 수록, 코드가 엄청 복잡해짐 ?? 해결방법 : 코드를 간결하고 가독성 높게 만들어주자

2) 가독성 X

3) 트러블 슈팅 X

→ 유지보수가 어렵다

cf. 유지보수란? 10년~ 운영! 요구사항 반영, 에러 해결

컨트롤러

- 프로젝트에서 매니저 역할을 하는 파일
- 누군가에게 일을 어떻게 시켜야할 지 알고 있습니다.
  = 일을 직접하지는 않을 것
- 라우터를 통해 "사용자의 요청이" 길(url)을 찾아오면

  매니저(콜백함수 = controller)가 환영해줌. → 알바생한테 일을 시키고, 결과물을 매니저에게 전달

  매니저가 사용자에게 res를 돌려줌

비밀번호 암호화

```
const crypto = require('crypto'); //설치 필요없음

const salt = crypto.randomBytes(64).toString('base64');
const hashPassword = crypto.pbkdf2Sync(password, salt, 10000,64,
```

- salt > crypto라는 모듈에는 64바이트 크기로 랜덤으로 만들어주는 게 있음. 이후 base64크기의 문자열로 변환해줌

- crypto.pbkdf2Sync: password based key derivation function 2의 약자로 해싱할 값, 어떻게 해싱할 지, 몇 번 해싱할지, 몇 바이트, 알고리즘 순서로 넣음

route와 controller 분리

users.js

```
const express = require("express");
const router = express.Router();
const conn = require('../mariadb');

const {join,login,PasswordResetRequest,PasswordReset} = require

router.use(express.json()); //json 형식으로 들어올 것이다.
//회원가입
router.post('/join', join);
//로그인
router.post('/login', login);
//비밀번호 초기화 요청
router.post('/reset', PasswordResetRequest);
//비밀번호 초기화
router.put('/reset', PasswordReset);

module.exports = router;
```

usercController.js

```
const conn = require('../mariadb');
const {StatusCodes} = require('http-status-codes');
const jwt = require('jsonwebtoken');
const dotenv = require('dotenv');
const crypto = require('crypto'); //암호화 담당 모듈
```

```
dotenv.config();

const join = (req, res) => {
    const {email, password} = req.body;

    //회원가입 시 비밀번호를 암호화해서 암호화된 비밀번호와, salt 값을 같이
    //로그인 시 이메일과 비밀번호(날 것) => salt값 꺼내서 비밀번호 암호화 
    let sql = 'INSERT INTO users (email, password, salt) VALUES
    const salt = crypto.randomBytes(10).toString('base64');
    const hashPassword = crypto.pbkdf2Sync(password, salt, 1000(

    let values = [email, hashPassword, salt];

    conn.query(sql, values,
        (err,results) => {
            if(err){
                console.log(err);
                return res.status(StatusCodes.BAD_REQUEST).end(
            }
            return res.status(StatusCodes.CREATED).json(results
        }
    )
};

const login = (req, res) =>{
    const {email,password} = req.body;
    let sql = 'SELECT * FROM users WHERE email = ?';

    conn.query(sql, email,
        (err,results) => {
            if(err){
                console.log(err);
                return res.status(StatusCodes.BAD_REQUEST).end(
            }

            const loginUser = results[0];
```

```
            //salt값 꺼내서 날 것으로 들어온 비밀번호를 암호화 해보고
            const hashPassword = crypto.pbkdf2Sync(password, sa
            //db 비밀번호 비교
            if(loginUser && loginUser.password == hashPassword)
                const token = jwt.sign({
                    email : loginUser.email,

                }, process.env.PRIVATE_KEY,{
                    expiresIn : '5m',
                    issuer : "soyeong"
                })

                //token 쿠키에 담기
                res.cookie("token", token,{
                    httpOnly : true
                })
                console.log(token)
                return res.status(StatusCodes.OK).json(results)
            }
            else{
                return res.status(StatusCodes.UNAUTHORIZED).end
            }
            return res.status(StatusCodes.CREATED).json(results
        }
    )

};
const PasswordResetRequest = (req, res) =>{
    const {email} = res.body;

    let sql = 'SELECT * FROM users WHERE email = ?';

    conn.query(sql, email,
        (err,results) => {
            if(err){
```

```
                console.log(err);
                return res.status(StatusCodes.BAD_REQUEST).end(
            }
            const user = result[0];
            if (user){
                return res.status(StatusCodes.OK).json({
                    email : email
                });
            }else{
                return res.status(StatusCodes.UNAUTHORIZED).end
            }
        }
    )
};
const PasswordReset = (req, res) =>{
    const {email, password} = res.body;

    let sql = 'UPDATE users SET password = ?, salt=? WHERE emai

    const salt = crypto.randomBytes(10).toString('base64');
    const hashPassword = crypto.pbkdf2Sync(password, salt, 1000
    let values = [password,email, salt]

    conn.query(sql, values,
        (err,results) => {
            if(err){
                console.log(err);
                return res.status(StatusCodes.BAD_REQUEST).end(
            }

            if (results.affectedRows == 0){
                return res.status(StatusCodes.BAD_REQUEST).end(
            }else{
                return res.status(StatusCodes.OK).json(results)
            }
        }
```

```
    )
};
module.exports = {
    join,
    login,
    PasswordResetRequest,
    PasswordReset
};
```