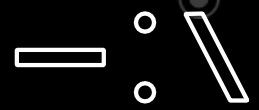
SHOUT OUR PASSION TOGETHER



PASSION

3rd Seminar

EXPERIENCE

GROWTH

CHALLENGE

SHOUT OUR PASSION TOGETHER

SOPT

조원

01 김강희, 김해리, 이다현, 이동훈, 이소희, 최소영

02 김채린, 박경선, 박시현, 양정훈, 양승희, 양희연, 조수민

03 김민준, 김정환, 남궁권, 이시연, 양시연, 제갈윤, 최영훈

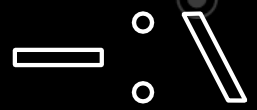
04 박승완, 심다은, 손예지, 윤혁, 이소연, 이재현, 장원준

05 강영우, 양희찬, 유희수, 이재용, 지현이, 조하담, 현주희

06 박주연, 박형모, 신윤재, 신정아, 허정민, 황채연

SHOUT OUR PASSION TOGETHER

목차



01

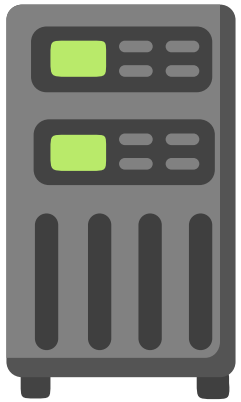
AWS 소개

Server란

서버(영어: server)는 클라이언트에게 네트워크를 통해 정보나 서비스를 제공하는 컴퓨터 시스템으로 **컴퓨터 프로그램**(server program) 또는 **장치**(device)를 의미한다. 특히, 서버에서 동작하는 소프트웨어를 **서버 소프트웨어**(server software)라 한다. 주로 리눅스 등의 운영 체제를 설치한 대형 컴퓨터를 쓰지만, 그렇지 않은 경우도 있다.

출처: 위키백과

Server란



서버

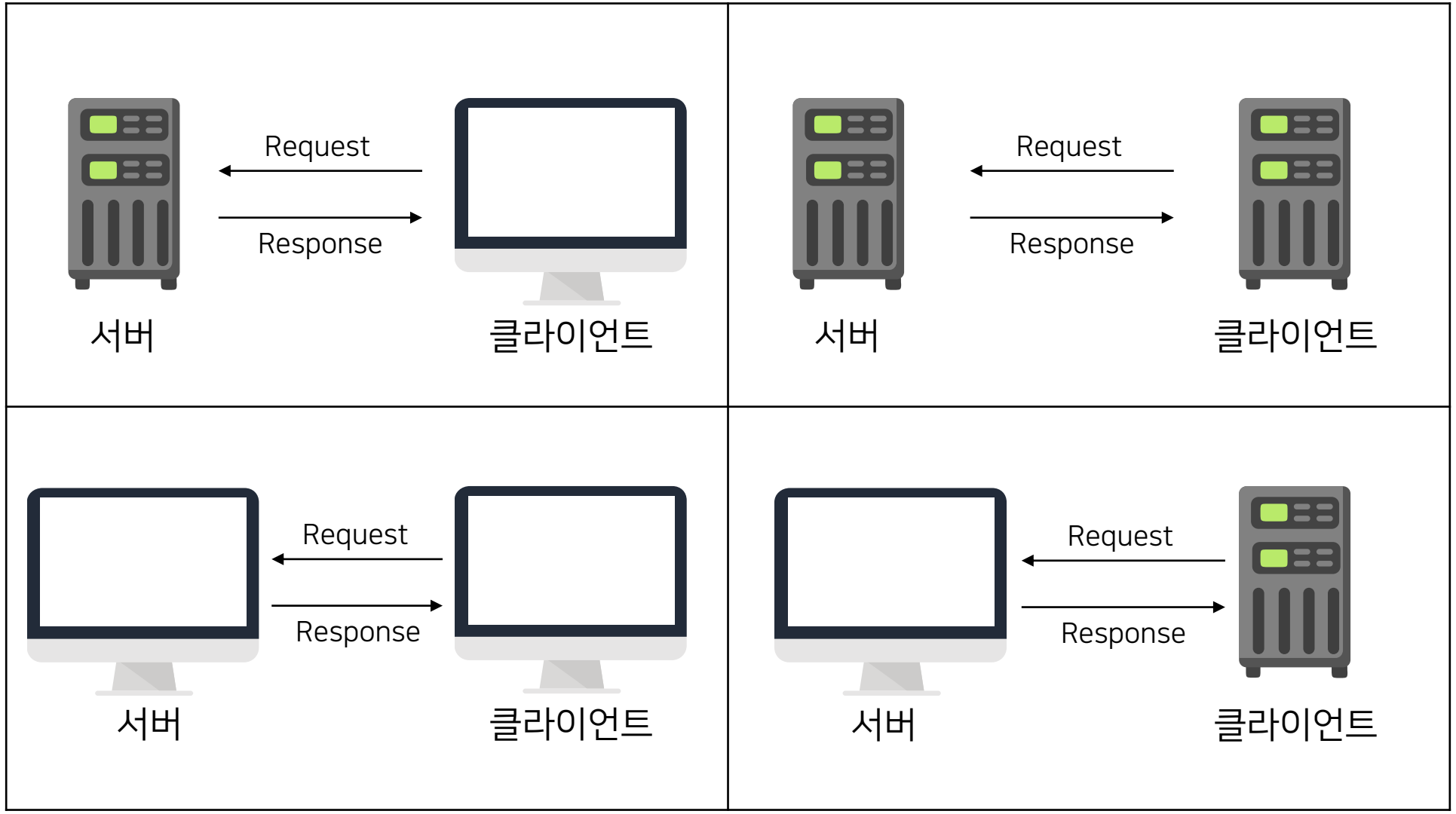
Request

Response

클라이언트

Server란

서버 컴퓨터도 데스크탑도 서버, 클라이언트가 될 수 있다.



Server의 조건

1. 컴퓨터 전원이 계속 ON되어 있어야한다.
2. Server Software가 특정 포트에 동작하고 있어야 한다.
3. Request가 들어오면 Response를 해야 한다.

Server의 조건

1. 컴퓨터 전원이 계속 ON되어 있어야한다.
2. Server Software가 특정 포트에 동작하고 있어야 한다.
3. Request가 들어오면 Response를 해야 한다.

- 서버 컴퓨터 구매 후 직접 구축
- 클라우드 플랫폼을 이용

클라우드 플랫폼 비교하기 https://itblogpro.tistory.com/150

Cloud Platform이란 클라우드 컴퓨팅은 인터넷(“클라우드”)을 통해

서버, 스토리지, 데이터베이스, 네트워킹,
소프트웨어, 분석, 인텔리전스 등의

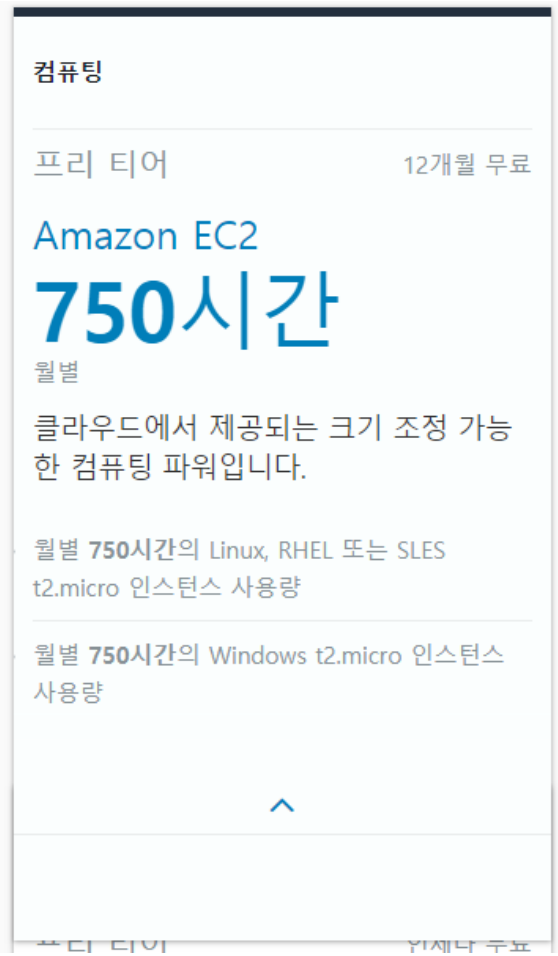
컴퓨팅 서비스를 제공하는 것입니다

AWS 소개

아마존 닷컴에서 개발한 클라우드 컴퓨팅 플랫폼
한 곳에서 IT 구축에 대한 모든 서비스 제공 받을 수 있음

저렴한 비용 저렴한 종량과금제 방식
즉각적 융통성 설치가 빠르고 관리가 편함
개방/유연성 언어 및 운영 체제에 구애 받지 않는 플랫폼
보안 여러 계층의 운영 및 물리적 보안 갖추고 있음

AWS 소개

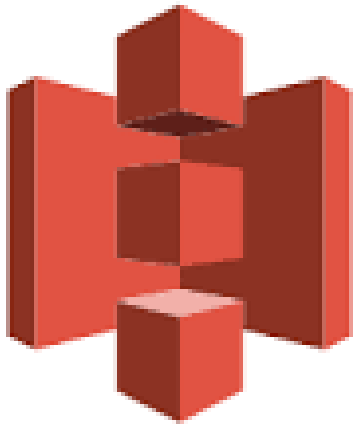


https://aws.amazon.com/ko/free/

Amazon EC2

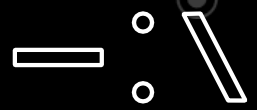
안전한 **크기 조정이 가능한**
컴퓨팅 파워를 클라우드에서 제공하는 웹 서비스
사용자가 정의한 조건에 따라 자동으로
Amazon EC2 용량을 급격하게 **확장 또는 축소** 기능

RDS는 아마존 웹 서비스가 서비스하는
분산 관계형 데이터베이스이다.
애플리케이션 내에서 관계형 데이터베이스의
설정, 운영, 스케일링을 단순하게 하도록 설계된
클라우드 내에서 동작하는 웹 서비스이다



amazon
S3

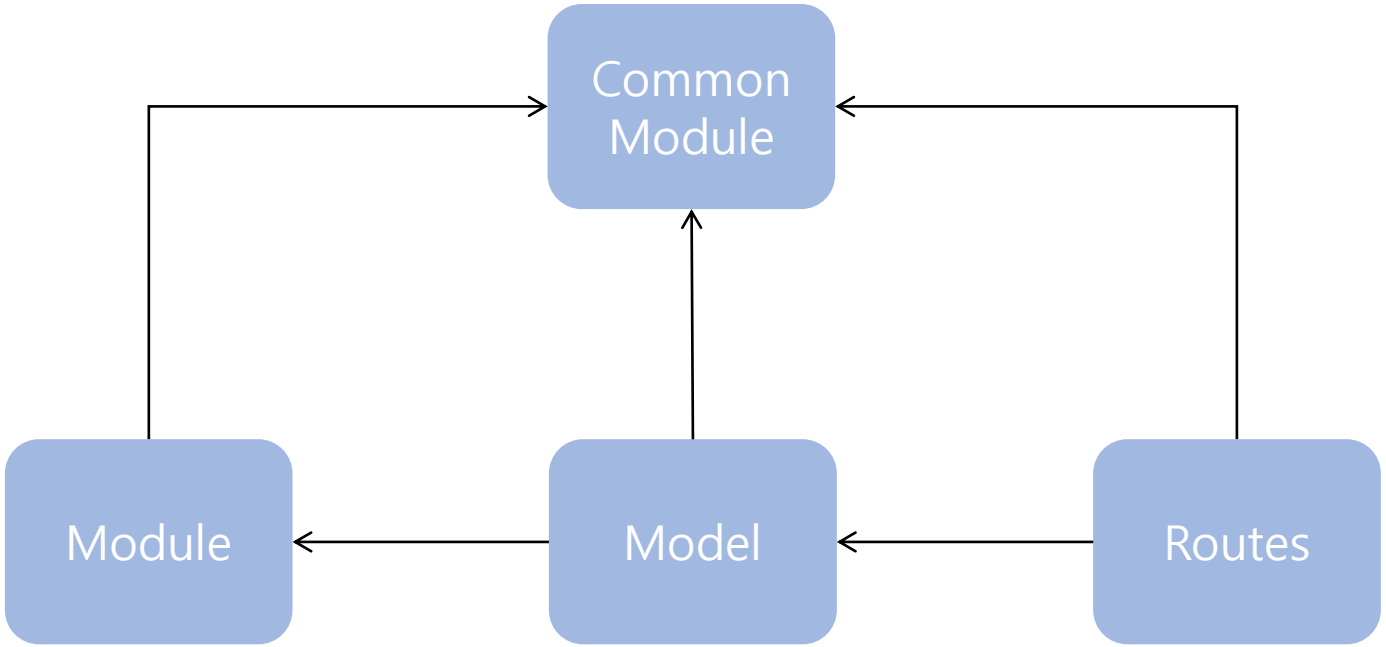
Amazon **S**imple **S**torage **S**ervice는
인터넷용 스토리지 서비스입니다.
이 서비스는 개발자가 더 쉽게
웹 규모 컴퓨팅 작업을 수행할 수 있도록 설계



02

CRUD 실습 (without DB)

프로젝트 구조



CommonModule:
공통되는 코드 및 상수 관리
(statusCode, responseMessage…)

Routes:
라우팅 로직 관리
(index.js…)

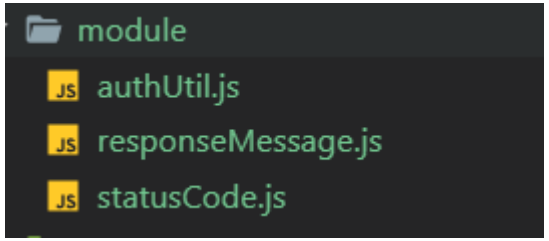
Model:
추상화 된 개념별로 로직 관리
(User, Board…)

Module:
기능 단위의 로직 관리
(암호화, DB매니저, csv매니저…)

1. Common Module 추가

Express 프로젝트 아래에 module 폴더를 만들고 아래의 파일을 추가한다.

1. authUtil.js
2. responseMessage.js
3. statusCode.js

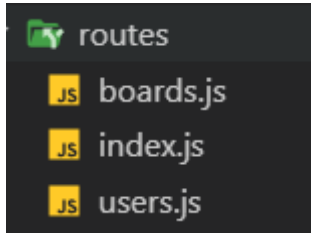



```
// authUtil.js
const authUtil = {
    successTrue: (message, data) => {
        return {
            success: true,
            message: message,
            data: data
        }
    },
    successFalse: (message) => {
        return {
            success: false,
            message: message
        }
    },
}
module.exports = authUtil
```

```
// responseMessage.js
module.exports = {
    NULL_VALUE: "필요한 값이 없습니다.",
    OUT_OF_VALUE: "파라미터 값이 잘못 되었습니다.",

    SIGN_UP_SUCCESS: "회원가입 성공",
    SIGN_UP_FAIL: "회원 가입 실패",
    SIGN_IN_SUCCESS: "로그인 성공",
    SIGN_IN_FAIL: "로그인 실패",
    ALREADY_ID: "존재하는 ID 입니다.",
    NO_USER: "존재하지 않는 유저 id 입니다.",
    MISS_MATCH_PW: "비밀번호가 일치하지 않습니다",


    BOARD_CREATE_SUCCESS: "게시글 작성 성공",
    BOARD_CREATE_FAIL: "게시글 작성 실패",
    BOARD_READ_ALL_SUCCESS: "게시글 전체 조회 성공",
    BOARD_READ_ALL_FAIL: "게시글 전체 조회 성공",
    BOARD_READ_SUCCESS: "게시글 조회 성공",
    BOARD_READ_FAIL: "게시글 조회 성공",
    BOARD_UPDATE_SUCCESS: "게시글 수정 성공",
    BOARD_UPDATE_FAIL: "게시글 수정 실패",
    BOARD_DELETE_SUCCESS: "게시글 삭제 성공",
    BOARD_DELETE_FAIL: "게시글 삭제 실패",

    ALREADY_ID: "존재하는 ID 입니다.",
    NO_USER: "존재하지 않는 유저 입니다.",
    NO_BOARD: "존재하는 게시글 입니다.",
    MISS_MATCH_PW: "비밀번호가 일치하지 않습니다",

    INTERNAL_SERVER_ERROR: "서버 내부 오류"
}
```

```
// statusCode.js
module.exports = {
    OK: 200,
    CREATED: 201,
    NO_CONTENT: 204,
    RESET_CONTENT: 205,
    NOT_MODIFIED: 304,
    BAD_REQUEST: 400,
    UNAUTHORIZED: 401,
    FORBIDDEN: 403,
    NOT_FOUND: 404,
    INTERNAL_SERVER_ERROR: 500,
    SERVICE_UNAVAILABLE: 503,
    DB_ERROR: 600,
}
```

2. Routes 구현



index.js

```
const express = require('express');
const router = express.Router();

router.use('/users', require('./users'));
router.use('/boards', require('./boards'));

module.exports = router;
```

user.js

```
const express = require('express');
const router = express.Router();

router.post('/signin', (req, res) => {
    // TODO 1: 파라미터 값 체크
    // TODO 2: 존재하는 아이디인지 확인
    // TODO 3: 비밀번호 일치하는지 확인
    // TODO 4: 유저 정보 응답하기
})

router.post('/signup', (req, res) => {
    // TODO 1: 파라미터 값 체크
    // TODO 2: 존재하는 ID인지 확인한다.
    // TODO 3: 사용자 정보를 저장한다.
    // TODO 4: 새로 추가된 유저 index 반환하기
})
module.exports = router;
```

board.js

```
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => { /* TODO : 게시글 전체 보기 */});
router.get('/:id', (req, res) => { /* TODO : 게시글 개별 보기  */});
router.post('/', (req, res) => { /* TODO : 게시글 작성 하기 });
router.put('/', (req, res) => { /* TODO : 게시글 수정 하기 });
router.delete('/', (req, res) => { /* TODO : 게시글 삭제 하기 });

module.exports = router;
```

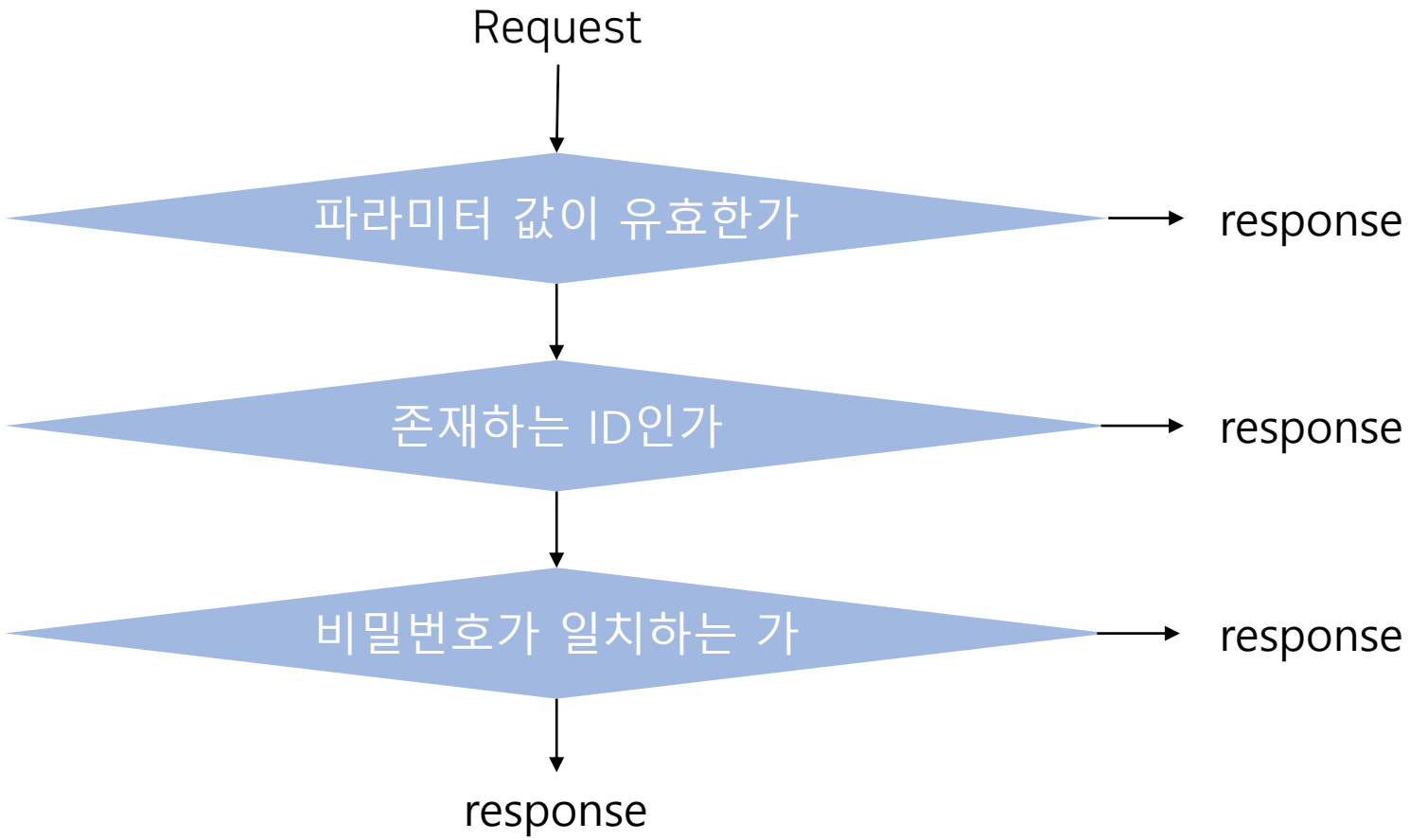
실습
회원가입, 로그인 구현 해보기

2. 회원가입, 실습 구현하기

```
const express = require('express');
const router = express.Router();
const statusCode = require('../module/statusCode');
const responseMessage = require('../module/responseMessage');
const authUtil = require('../module/authUtil')

// database 연동 전에 메모리에서 사용자 정보 관리
const infoMap = [{
    id: 'sopt',
    pwd: '1234',
    name: 'sopt',
    phone: '010-2081-3818'
}, {
    id: 'heesung',
    pwd: 'hello',
    name: '희성',
    phone: '010-2081-3818'
}];
```

```
router.post('/signin', (req, res) => {
    const {
        id,
        pwd
    } = req.body;
    console.log(id, pwd);
    // TODO 1: 파라미터 값 체크
    if (!id || !pwd) {
        res.status(statusCode.BAD_REQUEST)
            .send(authUtil.successFalse(responseMessage.NULL_VALUE));
        return;
    }
    // TODO 2: 존재하는 아이디인지 확인 (실패시 400 Error)
    const arr = infoMap.filter(it => it.id == id);
    if (arr.length == 0) {
        res.status(statusCode.BAD_REQUEST)
            .send(authUtil.successFalse(responseMessage.NO_USER));
        return;
    }
    // TODO 3: 비밀번호 일치하는지 확인 (실패시 401 Error)
    const user = arr[0];
    if (user.pwd != pwd) {
        res.status(statusCode.UNAUTHORIZED)
            .send(authUtil.successFalse(responseMessage.MISS_MATCH_PW));
        return;
    }
    // TODO 4: 유저 정보 응답하기
    res.status(statusCode.OK)
        .send(authUtil.successTrue(responseMessage.SIGN_IN_SUCCESS, user));
})
```

```
router.post('/signup', (req, res) => {
    const {
        id,
        pwd,
        name,
        phone
    } = req.body;
    console.log(id, pwd, name, phone)
    // TODO 1: 파라미터 값 체크
    if (!id || !pwd || !pwd || !name || !phone) {
        res.status(statusCode.BAD_REQUEST)
            .send(authUtil.successFalse(responseMessage.NULL_VALUE));
        return;
    }
    // TODO 2: 존재하는 ID인지 확인한다. (실패시 401 Error)
    if (infoMap.filter(it => it.id == id).length > 0) {
        res.status(statusCode.UNAUTHORIZED)
            .send(authUtil.successFalse(responseMessage.ALREADY_ID));
        return;
    }
    // TODO 3: 사용자 정보를 저장한다.
    const userIdx = infoMap.push({
        id,
        pwd,
        name,
        phone
    });
    console.log(infoMap);
    // TODO 4: 새로 추가된 유저 index 반환하기
    res.status(statusCode.OK)
        .send(authUtil.successTrue(responseMessage.SIGN_UP_SUCCESS, userIdx));
})
module.exports = router;
```


Request
파라미터 값이 유효한가
response
존재하는 ID인가
response
비밀번호가 일치하는 가
response
response

HTTP 실습

3. Model 폴더 생성 후 User.js 생성

routes에서 구현한 로그인, 회원가입 로직을 model로 분리한다.

```
const statusCode = require('../module/statusCode');
const responseMessage = require('../module/responseMessage');
const authUtil = require('../module/authUtil')

/*
    아래 infoMap은 DB에 적용하기 이전에 임시 변수입니다.
    즉 require 요청한 블록에 생성됩니다.
*/
const infoMap = [{
    id: 'sopt',
    pwd: '1234',
    name: 'sopt',
    phone: '010-2081-3818'
}, {
    id: 'heesung',
    pwd: 'hello',
    name: '희성',
    phone: '010-2081-3818'
}];
```

```
const user = {
    signin: (id, pwd) => {
        return new Promise((resolve, reject) => {
            // TODO 2: 존재하는 아이디인지 확인 (실패시 400 Error)
            const arr = infoMap.filter(it => it.id == id);
            if (arr.length == 0) {
                resolve({
                    code: statusCode.BAD_REQUEST,
                    json: authUtil.successFalse(responseMessage.NO_USER)
                });
                return;
            }
            // TODO 3: 비밀번호 일치하는지 확인 (실패시 401 Error)
            const user = arr[0];
            if (user.pwd != pwd) {
                resolve({
                    code: statusCode.UNAUTHORIZED,
                    json: authUtil.successFalse(responseMessage.MISS_MATCH_PW)
                });
                return;
            }
            // TODO 4: 유저 정보 응답하기
            resolve({
                code: statusCode.OK,
                json: authUtil.successTrue(responseMessage.SIGN_IN_SUCCESS, user)
            });
        });
    },
    signup: (id, pwd, name, phone) => {
        return new Promise((resolve, reject) => {
            // TODO 2: 존재하는 ID인지 확인한다. (실패시 401 Error)
            if (infoMap.filter(it => it.id == id).length > 0) {
                resolve({
                    code: statusCode.UNAUTHORIZED,
                    json: authUtil.successFalse(responseMessage.ALREADY_ID)
                });
                return;
            }
            // TODO 3: 사용자 정보를 저장한다.
            const userIdx = infoMap.push({
                id,
                pwd,
                name,
                phone
            });
            console.log(infoMap);
            // TODO 4: 새로 추가된 유저 index 반환하기
            resolve({
                code: statusCode.OK,
                json: authUtil.successTrue(responseMessage.SIGN_UP_SUCCESS, userIdx)
            });
        });
    }
}
module.exports = user
```

SHOUT OUR PASSION TOGETHER SOPT

https://github.com/WITH-SOPT-SERVER/SOPT-SERVER-SEMINAR/blob/develop/seminar3/httpPracticeProject/model/user.js

4. routes/user.js 수정

```
…
const User = require('../model/user');
…

router.post('/signin', (req, res) => {
    const {
        id,
        pwd
    } = req.body;
    console.log(id, pwd);
    // TODO 1: 파라미터 값 체크
    if (!id || !pwd) {
        res.status(statusCode.BAD_REQUEST)
            .send(authUtil.successFalse(responseMessage.NULL_VALUE));
        return;
    }
    User.signin(id, pwd)
        .then(({code, json}) => res.status(code).send(json))
        .catch(err => {
            console.log(err);
            res.status(statusCode.INTERNAL_SERVER_ERROR,
authUtil.successFalse(responseMessage.INTERNAL_SERVER_ERROR))
        });
})
```

```
router.post('/signup', (req, res) => {
    const {
        id,
        pwd,
        name,
        phone
    } = req.body;
    console.log(id, pwd, name, phone)
    // TODO 1: 파라미터 값 체크
    if (!id || !pwd || !pwd || !name || !phone) {
        res.status(statusCode.BAD_REQUEST)
            .send(authUtil.successFalse(responseMessage.NULL_VALUE));
        return;
    }

    User.signup(id, pwd, name, phone)
    .then(({code, json}) => res.status(code).send(json))
    .catch(err => {
        console.log(err);
        res.status(statusCode.INTERNAL_SERVER_ERROR,
 authUtil.successFalse(responseMessage.INTERNAL_SERVER_ERROR))
    })
})
module.exports = router;
```

SHOUT OUR PASSION TOGETHER SOPT

https://github.com/WITH-SOPT-SERVER/SOPT-SERVER-SEMINAR/blob/develop/seminar3/httpPracticeProject/routes/users.js

유저 회원가입, 로그인을 참고해서
게시판 CRUD를 만들어 보다

HTTP 실습

5. model/board.js 작성

```
const statusCode = require('../module/statusCode');
const responseMessage = require('../module/responseMessage');
const authUtil = require('../module/authUtil')

const boardArr = [{
    title: 'sopt',
    content: 'hello',
    writer: '솝트',
    pwd: '1234',
    time: Date.now(),
}, {
    title: 'heesung',
    content: 'hello',
    writer: '희성',
    pwd: '1234',
    time: Date.now()
}];

const board = {
    create: (title, content, writer, pwd) => {
        return new Promise((resolve, reject) => {
            const idx = boardArr.push({
                title,
                content,
                writer,
                pwd,
                time: Date.now()
            });
            resolve({
                code: statusCode.OK,
                json: authUtil.successTrue(
                    responseMessage.BOARD_CREATE_SUCCESS,
                    idx
                )});
        });
    },
    readAll: () => {
        return new Promise((resolve, reject) => {
            resolve({
                code: statusCode.OK,
                json: authUtil.successTrue(
                    responseMessage.BOARD_READ_ALL_SUCCESS,
                    boardArr
                )});
        });
    },
    read: (idx) => {
        return new Promise((resolve, reject) => {
            if (idx >= boardArr.length) {
                resolve({
                    code: statusCode.BAD_REQUEST,
                    json: authUtil.successFalse(responseMessage.NO
_BOARD)
                });
                return;
            }
            resolve({
                code: statusCode.OK,
                json: authUtil.successTrue(
                    responseMessage.BOARD_READ_ALL_SUCCESS,
                    boardArr[idx])
                });
        });
    },
    update: (idx, title, content, writer, pwd) => {
        return new Promise((resolve, reject) => {
            // idx값 확인
            if (idx >= boardArr.length) {
                resolve({
                    code: statusCode.BAD_REQUEST,
                    json: authUtil.successFalse(responseMessage.NO_BOARD)
                });
                return;
            }
            // 비밀번호 확인
            if (boardArr[idx].pwd != pwd) {
                resolve({
                    code: statusCode.FORBIDDEN,
                    json: authUtil.successFalse(responseMessage.MISS_MATCH_PW)
                });
                return;
            }
            boardArr[idx].title = title;
            boardArr[idx].content = content;
            boardArr[idx].writer = writer;
            resolve({
                code: statusCode.OK,
                json: authUtil.successTrue(responseMessage.BOARD_UPDATE_SUCCESS, boardArr[idx])
            });
        });
    },
    delete: (idx, pwd) => {
        return new Promise((resolve, reject) => {
            // idx값 확인
            if (idx >= boardArr.length) {
                resolve({
                    code: statusCode.BAD_REQUEST,
                    json: authUtil.successFalse(responseMessage.NO_BOARD)
                });
                return;
            }
            // 비밀번호 확인
            if (boardArr[idx].pwd != pwd) {
                resolve({
                    code: statusCode.FORBIDDEN,
                    json: authUtil.successFalse(responseMessage.MISS_MATCH_PW)
                });
                return;
            }
            boardArr[idx] = {};
            resolve({
                code: statusCode.OK,
                json: authUtil.successTrue(responseMessage.BOARD_DELETE_SUCCESS)
            });
        });
    }
}
module.exports = board
```

SHOUT OUR PASSION TOGETHER SOPT

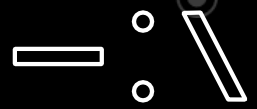
https://github.com/WITH-SOPT-SERVER/SOPT-SERVER-SEMINAR/blob/develop/seminar3/httpPracticeProject/model/board.js

HTTP 실습

5. routes/board.js 작성

```
const express = require('express');
const router = express.Router();
const statusCode = require('../module/statusCode');
const responseMessage = require('../module/responseMessage');
const authUtil = require('../module/authUtil');
const Board = require('../model/board');

router.get('/', (req, res) => {
    Board.readAll().then(({
        code,
        json
    }) => {
        res.status(code).send(json);
    }).catch(err => {
        console.log(err);
        res.status(statusCode.INTERNAL_SERVER_ERROR, authUtil.successFalse(responseMessage.INTERNAL_SERVER_ERROR));
    });
})

router.get('/:id', (req, res) => {
    const id = req.params.id;
    if (!id) {
        res.status(statusCode.BAD_REQUEST,
            authUtil.successFalse(responseMessage.NULL_VALUE));
        return;
    }
    Board.read(id).then(({
        code,
        json
    }) => {
        res.status(code).send(json);
    }).catch(err => {
        console.log(err);
        res.status(statusCode.INTERNAL_SERVER_ERROR, authUtil.successFalse(responseMessage.INTERNAL_SERVER_ERROR));
    });
})

router.post('/', (req, res) => {
    const {
        title,
        content,
        writer,
        pwd,
    } = req.body;

    if (!title || !content || !writer || !pwd) {
        res.status(statusCode.BAD_REQUEST,
            authUtil.successFalse(responseMessage.NULL_VALUE));
        return;
    }

    Board.create(title,content, writer, pwd).then(({
        code,
        json
    }) => {
        res.status(code).send(json);
    }).catch(err => {
        console.log(err);
        res.status(statusCode.INTERNAL_SERVER_ERROR, authUtil.successFalse(responseMessage.INTERNAL_SERVER_ERROR));
    });
})

router.put('/', (req, res) => {
    const {
        idx,
        title,
        content,
        writer,
        pwd
    } = req.body;
    if (!title || !content || !content || !writer || !pwd) {
        res.status(statusCode.BAD_REQUEST,
            authUtil.successFalse(responseMessage.NULL_VALUE));
        return;
    }

    Board.update(idx, title,content, writer, pwd).then(({
        code,
        json
    }) => {
        res.status(code).send(json);
    }).catch(err => {
        console.log(err);
        res.status(statusCode.INTERNAL_SERVER_ERROR, authUtil.successFalse(responseMessage.INTERNAL_SERVER_ERROR));
    });
})

router.delete('/', (req, res) => {
    const {
        idx,
        pwd
    } = req.body;
    if (!idx || !pwd) {
        res.status(statusCode.BAD_REQUEST,
            authUtil.successFalse(responseMessage.NULL_VALUE));
        return;
    }

    Board.delete(idx, pwd).then(({
        code,
        json
    }) => {
        res.status(code).send(json);
    }).catch(err => {
        console.log(err);
        res.status(statusCode.INTERNAL_SERVER_ERROR, authUtil.successFalse(responseMessage.INTERNAL_SERVER_ERROR));
    });
})

module.exports = router;
```


https://github.com/WITH-SOPT-SERVER/SOPT-SERVER-SEMINAR/blob/develop/seminar3/httpPracticeProject/model/board.js



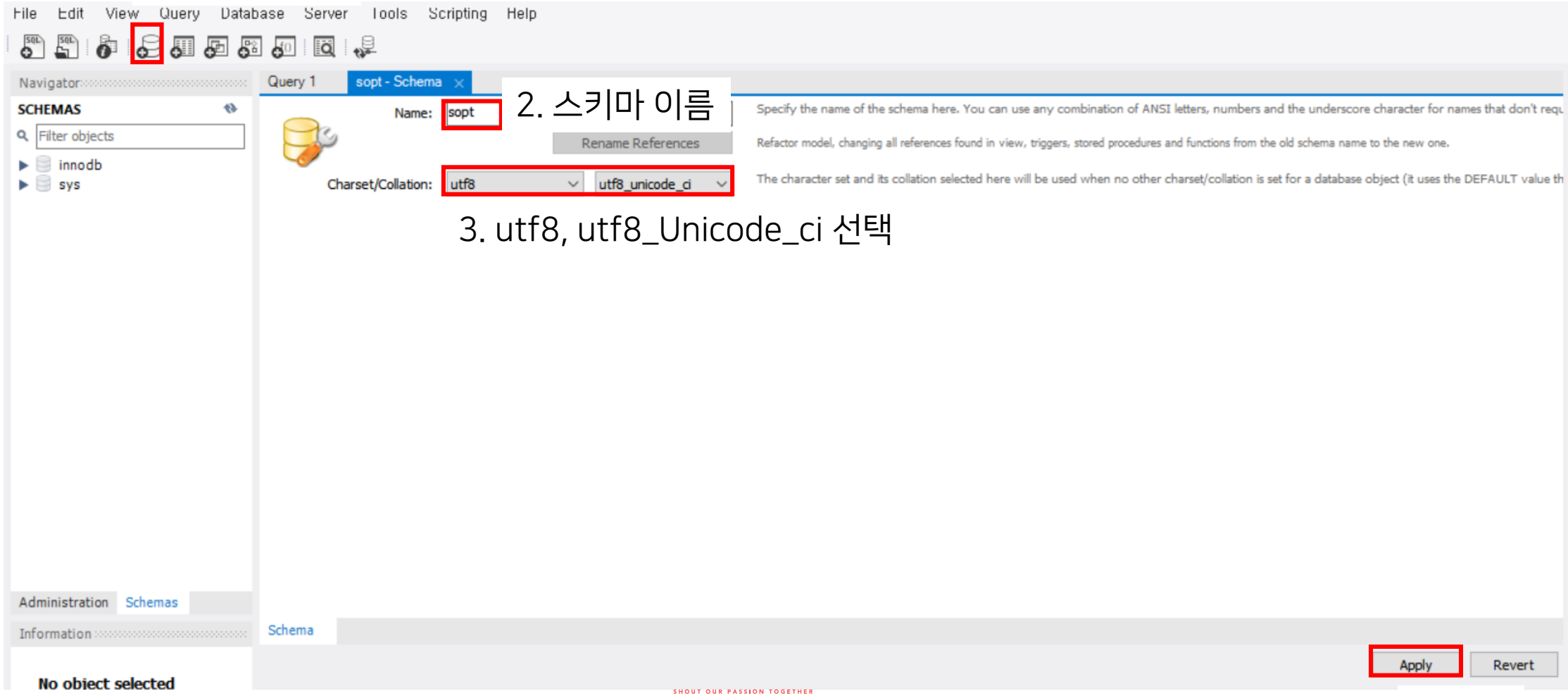
03

RDS 연동

RDS 연동

스키마 생성

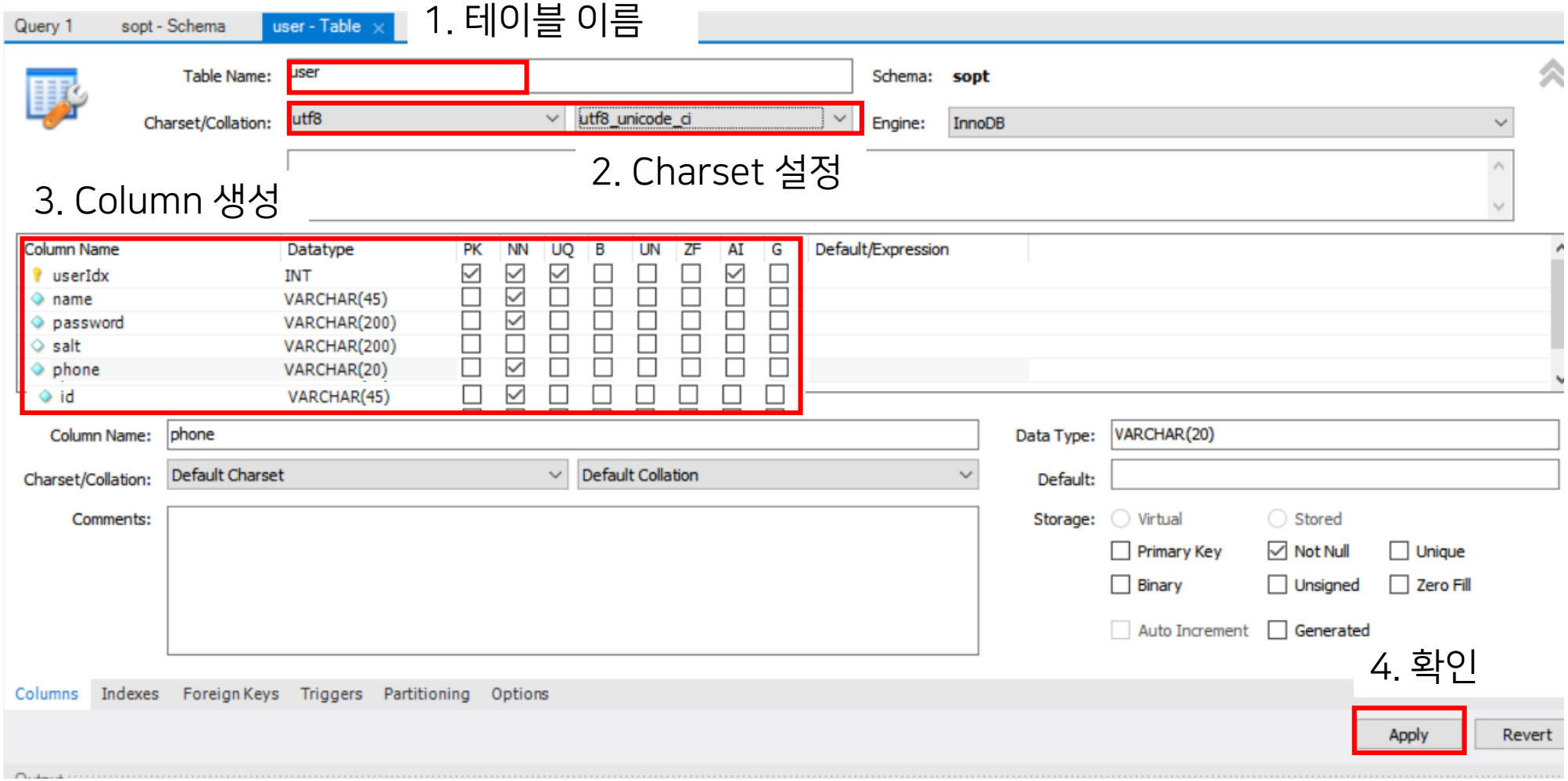
1. 스키마 생성



4. 확인

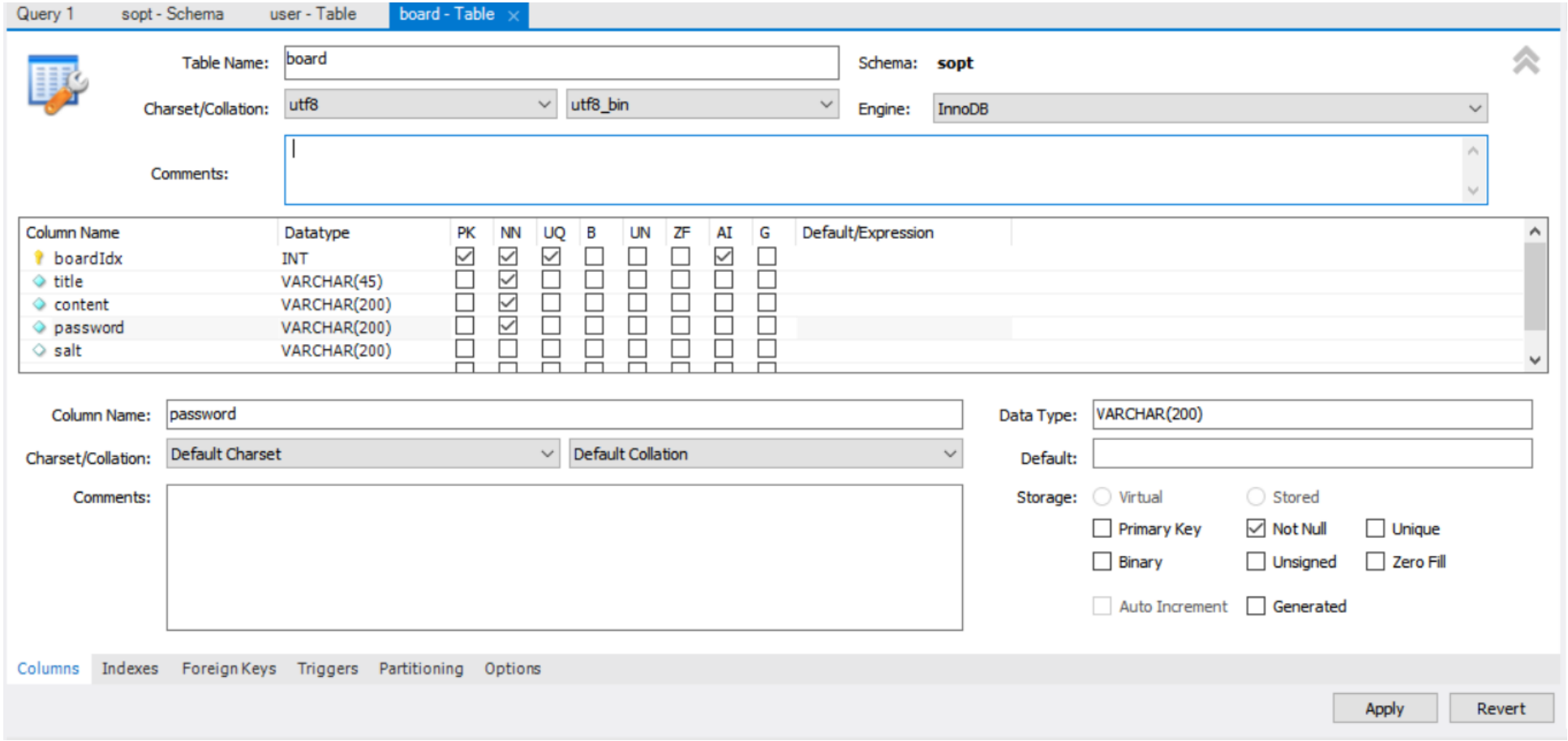
RDS 연동

User 테이블 생성



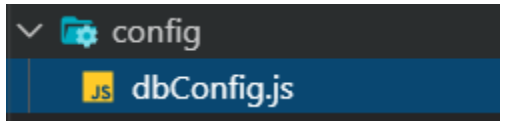
RDS 연동

Board 테이블 생성



RDS 연동

1. Config 폴더 생성 및 파일 추가



2. .gitignore에 아래 내용 추가

```
# config 파일
config/*
dbConfig.js
```

3. dbConfig.js에 아래 내용 추가

```
const mysql = require('promise-mysql')

const dbConfig = {
    host: 'db-with-sopt-server.~~~~~~.ap-northeast-
2.rds.amazonaws.com',
    port: 3306,
    user: 'admin',
    password: 'RDS 패스워드 ' ,
    database: 'RDS DB 이름',
    dateStrings: 'date',
}

module.exports = mysql.createPool(dbConfig)
```

RDS 연동

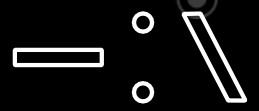
4. Module 폴더에 pool.js 추가. (아래 코드는 4차 세미나에서 다룰 예정입니다.)

```
const poolPromise = require('../config/dbConfig')

module.exports = {
    queryParam_None: async (...args) => {
        const query = args[0]
        let result
        const pool = await poolPromise;
        try {
            var connection = await pool.getConnection() // connection을 pool에서 하나 가져온다.
            result = await connection.query(query) ¦¦ null // query문의 결과 ¦¦ null 값이 result에 들어간다.
        } catch (err) {
            console.log(err)
            connection.rollback(() => {})
        } finally {
            pool.releaseConnection(connection) // waterfall 에서는 connection.release()를 사용했지만, 이 경우 pool.releaseConnection(connection) 을 해준다.
            return result
        }
    },
    queryParam_Arr: async (...args) => {
        const query = args[0]
        const value = args[1] // array
        let result
        try {
            var connection = await pool.getConnection() // connection을 pool에서 하나 가져온다.
            result = await connection.query(query, value) ¦¦ null // 두 번째 parameter에 배열 => query문에 들어갈 runtime 시 결정될 value
        } catch (err) {
            connection.rollback(() => {})
            next(err)
        } finally {
            pool.releaseConnection(connection) // waterfall 에서는 connection.release()를 사용했지만, 이 경우 pool.releaseConnection(connection) 을 해준다.
            return result
        }
    },
    queryParam_Parse: async (inputquery, inputvalue) => {
        const query = inputquery
        const value = inputvalue
        let result
        try {
            var connection = await pool.getConnection()
            result = await connection.query(query, value) ¦¦ null
            console.log(result)
        } catch (err) {
            console.log(err)
            connection.rollback(() => {})
            next(err)
        } finally {
            pool.releaseConnection(connection)
            return result
        }
    },
    Transaction: async (...args) => {
        let result = "Success"

        try {
            var connection = await pool.getConnection()
            await connection.beginTransaction()
            await args[0](connection, ...args)
            await connection.commit()
        } catch (err) {
            await connection.rollback()
            console.log("mysql error! err log =>" + err)
            result = undefined
        } finally {
            pool.releaseConnection(connection)
            return result
        }
    }
}
```

RDS 연동

5. routes/dbTest.js 생성 이후 아래와 같이 입력 및 라우팅 연결

```
const express = require('express');
const router = express.Router();
const pool = require('../module/pool');

router.get('/insert', async (req, res) => {
    const table = 'user';
    const fields = 'id, name, password, phone';
    const questions = ``'sopt', 'sopt', '1234', '010-0000-0000'``;
    const result = await pool.queryParam_None(`INSERT INTO ${table}(${fields})
 VALUES(${questions})`)
    if(!result) {
        res.status(500).send('error');
        return;
    }
    res.status(200).send(result);
})

router.get('/select', async (req, res) => {
    const table = 'user';
    const result = await pool.queryParam_None(`SELECT * FROM ${table}`);
    if(!result) {
        res.status(500).send('error');
        return;
    }
    res.status(200).send(result);
})

router.get('/update', async (req, res) => {
    const table = 'user';
    const result = await pool.queryParam_None(`UPDATE ${table} SET name = 'so
ptTest' where name = 'sopt'`);
    console.log(result);
    if(!result) {
        res.status(500).send('error');
        return;
    }
    res.status(200).send(result);
})

router.get('/delete', async (req, res) => {
    const table = 'user';
    const result = await pool.queryParam_None(`DELETE FROM ${table} WHERE NAM
E='sopt'``)
    console.log(result);
    if(!result) {
        res.status(500).send('error');
        return;
    }
    res.status(200).send(result);
})

module.exports = router;
```



04

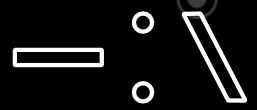
CRUD 실습 (with DB)

CRUD 실습

실습:
Model에 있는 파일을 수정해서
Database에 연동시켜주세요!!

EC2에 업로드 실습

1. Git clone repository
2. Config 파일 복사 붙여넣기
3. npm install
4. npm start



05

과제안내

과제 안내

Level1:

Blog 테이블을 만들고(칼럼은 각자 생각)
CRUD를 구현할 것.
[GET]/blogs [POST]/blogs [PUT]/blogs [DELETE]/blogs

Level2:

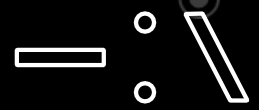
Article 테이블을 만들고(칼럼은 각자 생각해서 구현할 것)
CRUD를 구현할 것.
이때 article에는 blogIdx 칼럼 포함
[GET]/blogs/${blogIdx}/articles
[POST]/blogs/${blogIdx}/articles
[PUT]/blogs/${blogIdx}/articles
[DELETE]/blogs/${blogIdx}/articles

Level3:

Comment 테이블을 만들고(칼럼은 각자 생각)
CRUD를 구현할 것.
이때 comment에는 articleIdx 칼럼 포함
[GET]/blogs/${blogIdx}/articles/${articleIdx}/comments
[POST]/blogs/${blogIdx}/articles/${articleIdx}/comments
[PUT]/blogs/${blogIdx}/articles/${articleIdx}/comments
[DELETE]/blogs/${blogIdx}/articles/${articleIdx}/comments

ec2 서버 주소 및 github repository 주소로 제출 (api 주소는 지켜주세요!)

SHOUT OUR PASSION TOGETHER



PASSION

Thank You :)

EXPERIENCE

GROWTH

CHALLENGE

SHOUT OUR PASSION TOGETHER

SOPT