**미션 목록 조회(진행중/ 진행완료)**

GET /Users/Misson

Query String: ? user-id

request body: x

request Header: asscessToken (String)

**구현**

**1. 라우팅**

// mission.route.js

import express from 'express'; //express 모듈 임포트

import { showMission } from '../controllers/mission.controller.js';

export const missionRouter = express.Router();

missionRouter.get('/mission/:flag',showMission);

원래는 showMission 함수를 통해 현재 미션들의 리스트를 나타내는 함수를 구현하고 싶었으나 DB 및 구현 방식이 어려울 것 같아 연결에 중점을 두고 만들었다

**2. 컨트롤러**

// mission.controller.js

import { status } from '../../config/response.status.js';

//응답 상태 코드 반환. sucess

import { CheckID } from '../services/mission.service.js';

import { response } from '../../config/response.js';

//response 함수 임포트. 응답함수

export const showMission = (req,res,next)=>{

    console.log("/mission/"+req.params.flag);

    return res.send(response(status.SUCCESS, CheckID(req.params.flag)));

}

**3. 서비스**

// mission.service.js

import { userIDResponseDTO } from "../dtos/mission.response.dto.js";

export function CheckID(id){

    return userIDResponseDTO(id);

}

**4. DTO**

// mission.response.dto.js

export const userIDResponseDTO =(data)=>{

    return {"user\_ID" : data};

}

userIDResponseDTO 함수는 받은 데이터를 user\_ID 필드를 가진 객체로 변환하여 반환

특정 데이터를 일관된 형식으로 클라이언트에 전달하기 위한 DTO(Data Transfer Object) 역할

**5. index.js**

// index.js

import express from 'express';

import { missionRouter } from './src/routes/mission.route.js';

const app = express();

const port = 3000;

// router setting

app.use('/user',missionRouter); //'User/Mission

app.listen(port, () => {

    console.log(`Example app listening on port ${port}`)

});

**결과**

텍스트, 스크린샷, 폰트, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

-> mission 뒤에 숫자에 따라 바뀌는 것을 볼 수 있다.

**비동기 에러 헨들링 수행**

**1. error.js, response.status.js 파일 재확인**

// config/error.js

export class BaseError extends Error {

    constructor(data){

        super(data.message);

        this.data = data;

    }

}

// response.status.js

import { StatusCodes } from "http-status-codes";

export const status = {

    // success

    SUCCESS: {status: StatusCodes.OK, "isSuccess": true, "code": 2000, "message": "SUCCESS!"},

    // error

    // common err

    INTERNAL\_SERVER\_ERROR: {status: StatusCodes.INTERNAL\_SERVER\_ERROR, "isSuccess": false, "code": "COMMON000", "message": "서버 에러, 관리자에게 문의 바랍니다." },

    BAD\_REQUEST: {status: StatusCodes.BAD\_REQUEST, "isSuccess": false, "code": "COMMON001", "message": "잘못된 요청입니다." },

    UNAUTHORIZED: {status: StatusCodes.UNAUTHORIZED, "isSuccess": false, "code": "COMMON002", "message": "권한이 잘못되었습니다." },

    METHOD\_NOT\_ALLOWED: {status: StatusCodes.METHOD\_NOT\_ALLOWED, "isSuccess": false, "code": "COMMON003", "message": "지원하지 않는 Http Method 입니다." },

    FORBIDDEN: {status: StatusCodes.FORBIDDEN, "isSuccess": false, "code": "COMMON004", "message": "금지된 요청입니다." },

    NOT\_FOUND : {status: StatusCodes.NOT\_FOUND, "isSuccess": false, "code": "COMMON005", "message": "요청한 페이지를 찾을 수 없습니다. 관리자에게 문의 바랍니다."},

    // member err

    MEMBER\_NOT\_FOUND: {status: StatusCodes.BAD\_REQUEST, "isSuccess": false, "code": "MEMBER4001", "message": "사용자가 없습니다."},

    NICKNAME\_NOT\_EXIST: {status: StatusCodes.BAD\_REQUEST, "isSuccess": false, "code": "MEMBER4002", "message": "닉네임은 필수입니다."},

    // article err

    ARTICLE\_NOT\_FOUND: {status: StatusCodes.NOT\_FOUND, "isSuccess": false, "code": "ARTICLE4001", "message": "게시글이 없습니다."}

};

실습에 사용했던 위 파일들을 다시 사용할 예정이다

**2. 컨트롤러 수정**

// mission.controller.js

import { status } from '../../config/response.status.js';

import { CheckID } from '../services/mission.service.js';

import { response } from '../../config/response.js';

import { BaseError } from '../../config/error.js';

export const showMission = async (req, res, next) => {

    try {

        console.log("/mission/" + req.params.flag);

        const result = await CheckID(req.params.flag);

        res.send(response(status.SUCCESS, result));

    } catch (error) {

        if (error instanceof BaseError) {

            res.status(error.data.status).send(response(error.data, null));

        } else {

            next(new BaseError(status.INTERNAL\_SERVER\_ERROR, error.message));

        }

    }

};

비동기 함수 구현을 위해 showMission 함수의 checkID를 async/await으로 호출한다.

성공하면 결과를 반환하고 에러가 발생하면 try/catch로 처리하여 적절한 에러 응답을 보낸다.

**3. 서비스 수정**

// mission.service.js

import { userIDResponseDTO } from "../dtos/mission.response.dto.js";

import { BaseError } from "../../config/error.js";

import { status } from "../../config/response.status.js";

export const CheckID = async (flag) => {

    return new Promise((resolve, reject) => {

        setTimeout(() => {

            if (flag!=0) {

                resolve(userIDResponseDTO(flag));

            } else {

                reject(new BaseError(status.BAD\_REQUEST, 'Invalid flag provided'));

            }

        }, 1000);

    });

};

CheckID 함수를 promise를 사용한 비동기 함수로 수정하였다.

flag가 0이 아니면 성공(resolve), 0이면 에러(reject)를 발생시킨다.

**4. index.js 수정**

// index.js

import express from 'express';

import { response } from './config/response.js';

import { status } from './config/response.status.js';

import { BaseError } from './config/error.js';

import { missionRouter } from './src/routes/mission.route.js';

const app = express();

const port = 3000;

// router setting

app.use('/user',missionRouter); //'User/Mission

app.use((req, res, next) => {

    const err = new BaseError(status.NOT\_FOUND);

    next(err);

});

// error handling

app.use((err, req, res, next) => {

    // 템플릿 엔진 변수 설정

    res.locals.message = err.message;

    // 개발환경이면 에러를 출력하고 아니면 출력하지 않기

    res.locals.error = process.env.NODE\_ENV !== 'production' ? err : {};

    res.status(err.data.status).send(response(err.data));

});

app.listen(port, () => {

    console.log(`Example app listening on port ${port}`)

});

텍스트, 스크린샷, 폰트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

**추가) result 값이 null이 출력되는 이유**

해당 문제가 발생하는 이유는 두 가지 코드의 동작 방식이 다르기 때문

CheckFlag 함수에서는 동기적으로 에러를 발생시키고 있으며, CheckID 함수에서는 비동기적으로 에러를 발생시키고 있으므로 이로 인해 에러 처리가 다르게 동작함

CheckFlag 함수에서 동기적으로 에러를 발생시킬 때는 함수가 호출되는 시점에 바로 에러가 발생하여 호출한 코드 블록으로 에러가 전달됨 동기적으로 발생한 에러는 일반적인 try-catch 블록에서 잡을 수 있다

반면에, CheckID 함수에서는 비동기적으로 에러를 발생시키고 있다. 즉, 프로미스를 사용하여 비동기적으로 에러를 발생시키기 때문에, 에러 처리는 async/await 또는 .then().catch()를 사용하여 처리해야 한다.

이 두 가지 방식의 차이점 때문에 에러가 발생했을 때 응답 구조가 다르게 나타날 수 있다. 동기적으로 에러가 발생하는 경우와 비동기적으로 에러가 발생하는 경우를 통일된 방식으로 처리하려면, 에러 핸들링 로직을 일관되게 유지해야 한다.

**추가2) AccessToken을 구현하지 않은 이유**

// mission.controller.js

import { status } from '../../config/response.status.js';

import { CheckID } from '../services/mission.service.js';

import { response } from '../../config/response.js';

import { BaseError } from '../../config/error.js';

export const showMission = async (req, res, next) => {

    try {

        console.log("/mission/" + req.params.flag);

          // accessToken 헤더 읽기

        const accessToken = req.headers.accesstoken;

        if (!accessToken) {

            throw new BaseError(status.UNAUTHORIZED, 'Access token is required');

        }

        console.log("Access Token: ", accessToken); // accessToken 값 확인을 위한 콘솔 출력

        const result = await CheckID(req.params.flag);

        res.send(response(status.SUCCESS, result));

    } catch (error) {

        if (error instanceof BaseError) {

            res.status(error.data.status).send(response(error.data, null));

        } else {

            next(new BaseError(status.INTERNAL\_SERVER\_ERROR, error.message));

        }

    }

};

위와 같이 express.js에서 accesstoken을 받아오기 위해 해당 헤더를 받아오는 로직을 추가했지만 현재 로그인이 구현되지 않아 실제로 주어지는 값이 없기 때문에 다음과 같은 에러가 났다. 따라서 따로 구현하지 않았다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명