

삼성 청년 SW 아카데미

Node.js

<알림>

본 강의는 삼성 청년 SW아카데미의 콘텐츠로
보안서약서에 의거하여
강의 내용을 어떠한 사유로도 임의로 복사,
촬영, 녹음, 복제, 보관, 전송하거나
허가 받지 않은 저장매체를
이용한 보관, 제3자에게 누설, 공개,
또는 사용하는 등의 행위를 금합니다.

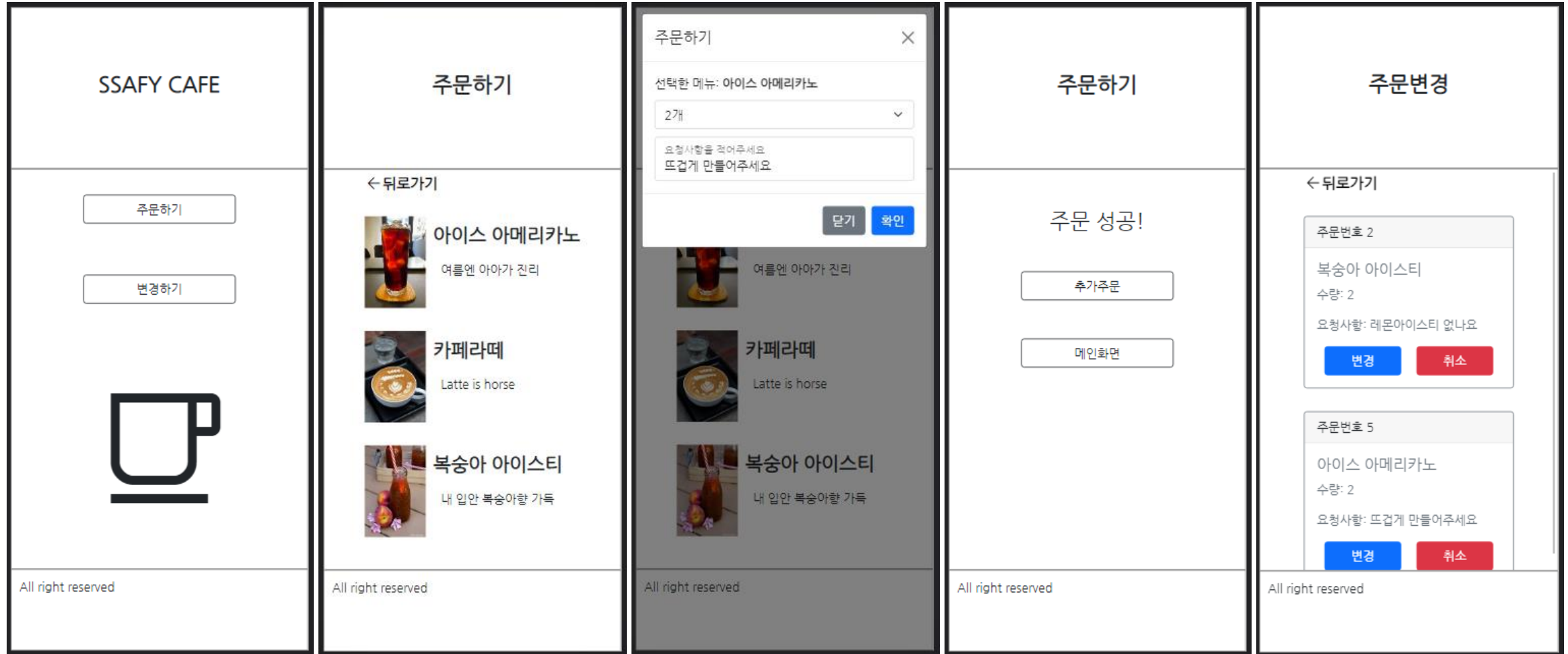
11장. REST API + MySQL

챕터의 포인트

- 카페 주문관리 API
- API 서버 배포

카페 주문관리 API

여러분은 다음과 같은 UI 를 가진, 카페 직원용 주문관리 시스템의 백엔드 파트를 담당하게 되었다.



요구 기능 분석

- 메뉴

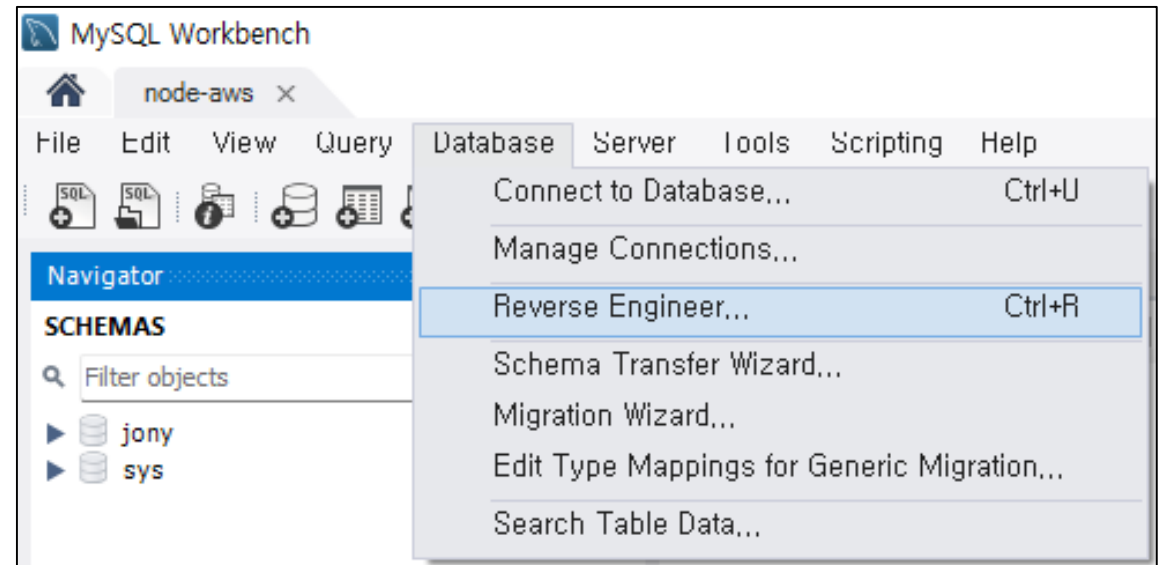
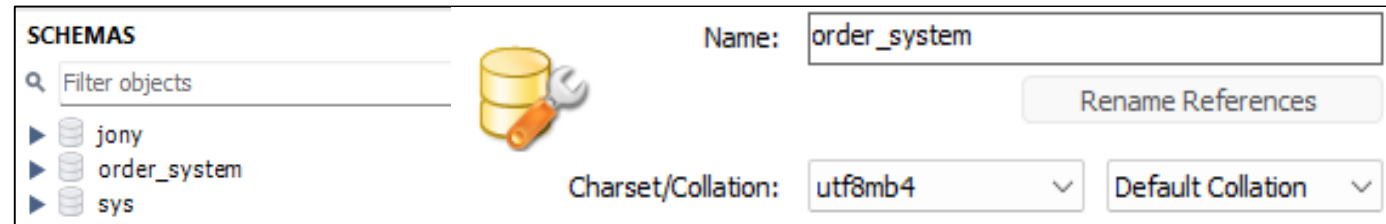
- 메뉴의 이름
- 메뉴의 정보
- 메뉴의 이미지 (파일 경로)

- 주문

- 주문할 메뉴
- 주문할 메뉴의 갯수
- 요청사항

workbench 접속

- mysql workbench 접속
- order_system 스키마 작성
 - utf8mb4 로 작성
- 메뉴 Database -> Reverse Engineer 접속



Reverse Engineer Database

- 다이어그램 형식을 활용해서 테이블 정의 및 관계형 설정이 가능
- 작성해둔 AWS DB서버에 접근
- 새로만든 order_system 스키마를 포함시키기

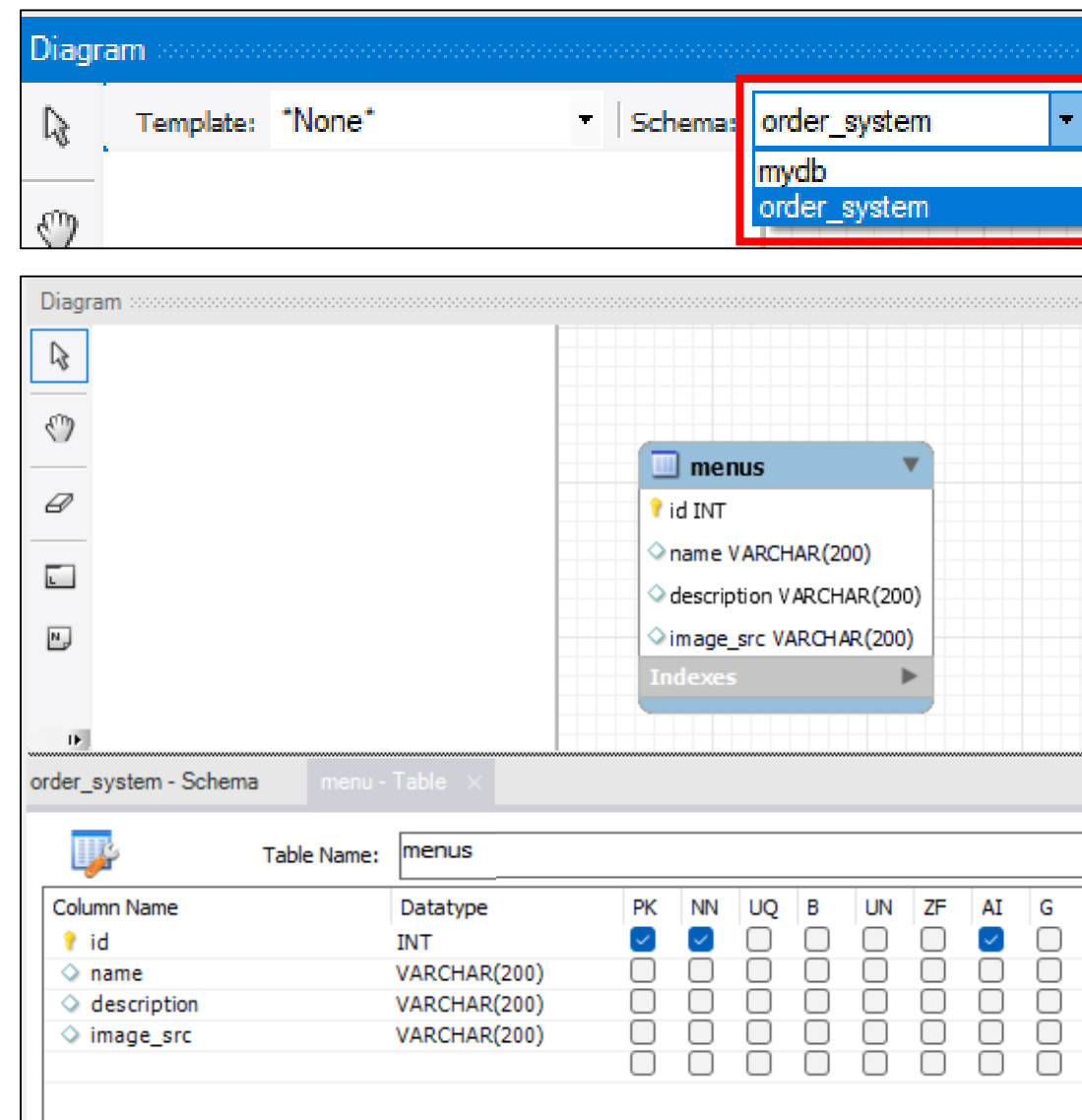
The screenshot shows the 'Reverse Engineer Database' application window. On the left is a sidebar with 'Connection Options' selected. The main area is titled 'Set Parameters for Connecting to a DBMS'. It contains fields for 'Stored Connection' (set to 'node-aws'), 'Connection Method' (set to 'Standard (TCP/IP)'), 'Hostname' (3.39.183.206), 'Port' (3306), 'Username' (ssafy), and 'Password' (with 'Store in Vault...' and 'Clear' buttons). A large blue arrow points from this window towards the right.

This dialog box is titled 'Select Schemas to Reverse Engineer'. It contains a database icon and the text 'Select the schemas you want to include:'. Below this, there are two checkboxes: 'jony' (unchecked) and 'order_system' (checked).

The 'Catalog Tree' shows a hierarchical view of the database. It lists 'mydb' and 'order_system'. Under 'order_system', there are three sub-items: 'Tables', 'Views', and 'Routine Groups'. The 'Tables' item is currently selected and highlighted.

- 메뉴 설계

- 테이블 클릭시 **스키마 선택**에 유의
- id
 - 고유 ID (중복 방지)
 - PK, NN, AI
- name
 - 메뉴의 이름
 - VARCHAR(200)
- description
 - VARCHAR(200)
- image_src
 - VARCHAR(200)



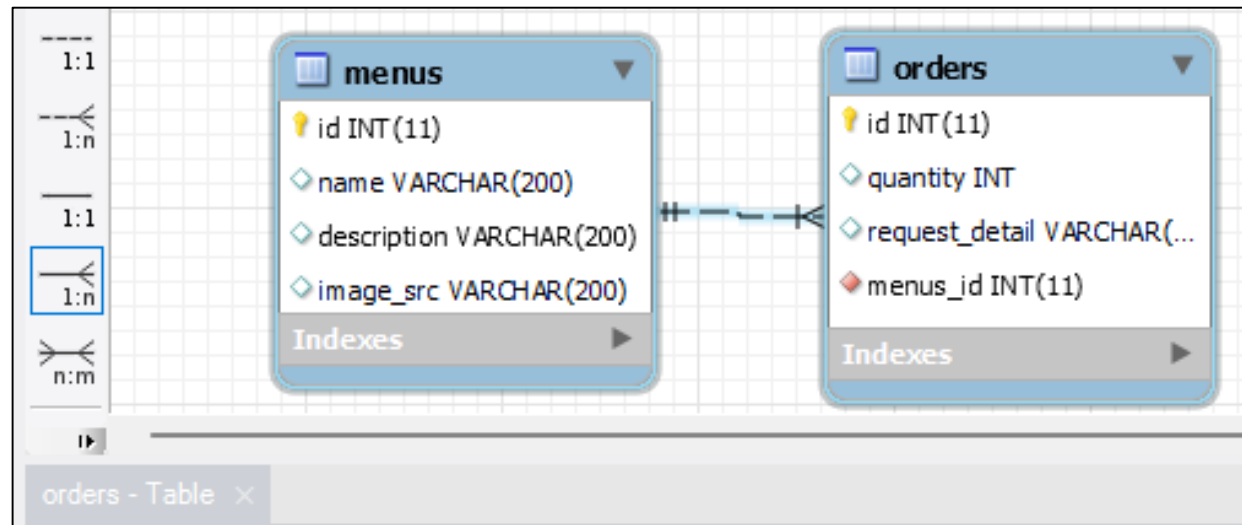
- **orders**

- id
 - 고유 ID (중복 방지)
 - PK, NN, AI
- quantity
 - 주문할 메뉴의 수
 - INT
- request_detail
 - 요청사항
 - VARCHAR(200)

[illegible]

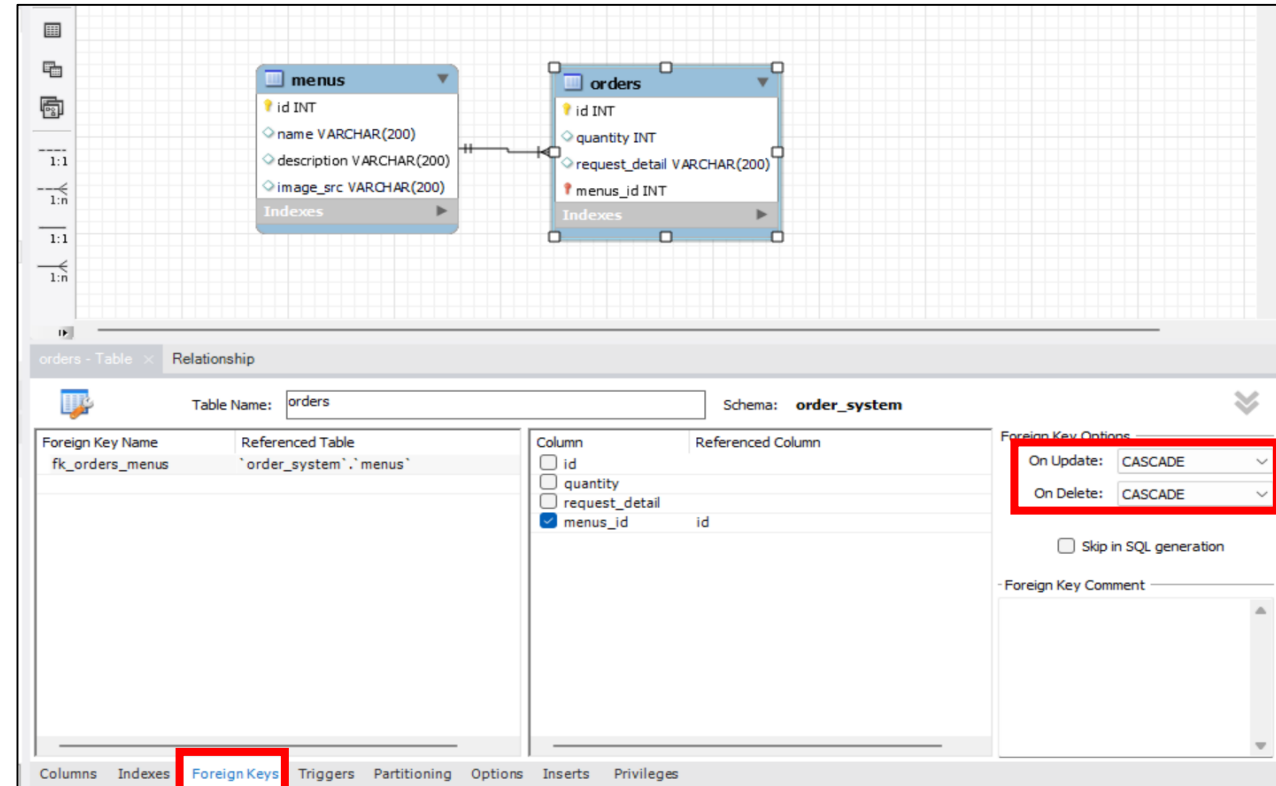
관계형 테이블 매핑

- orders에서는 menu 테이블의 정보가 필요
 - 메뉴의 정보는 menu 테이블에 존재한다.
 - id는 고유값이기 때문에 menu의 id를 통해 끌어오는 작업이 필요
 - order 테이블에서 menu_id 는 외래키가 된다.
 - 1:N 클릭 -> orders 클릭 후 menu 클릭
 - 주문할때 orders는 여러 개의 menu를 가질수 있다.



CASCADE 옵션 부여

- 외래키에 삭제/업데이트시 CASCADE 옵션부여
- orders 에는 menu_id가 존재한다.
 - 관련된 menu 삭제시에 외래키 조건에 의해 에러가 발생
 - CASCADE 옵션을 부여하면 menu 삭제시 menu 뿐만 아니라 해당 menu_id가 들어있는 모든 데이터를 삭제한다.



```
Error: Cannot delete or update a parent row: a foreign key constraint fails (`order_system`.`orders`, CONSTRAINT `fk_orders_menus` FOREIGN KEY (`menu_id`) REFERENCES `menu` (`id`) ON DELETE NO ACTION ON UPDATE NO ACTION)
```

CASCADE 미 부여상태로 관계된 테이블 데이터를 지울때 나오는 에러

• 1:1 관계

- 각 관계는 반드시 하나로만 매핑되어야한다.
 - ex) 남자 - 결혼 - 여자



• 1:N 관계

- 한쪽 개체는 다른 관계를 맺은쪽의 여러 개체를 가질 수 있다.
 - ex) 유저(users)는 게시글(posts)을 여러 개 작성 할 수 있다.
 - 부모는 여러명의 자식을 가질수 있다.



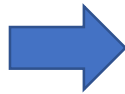
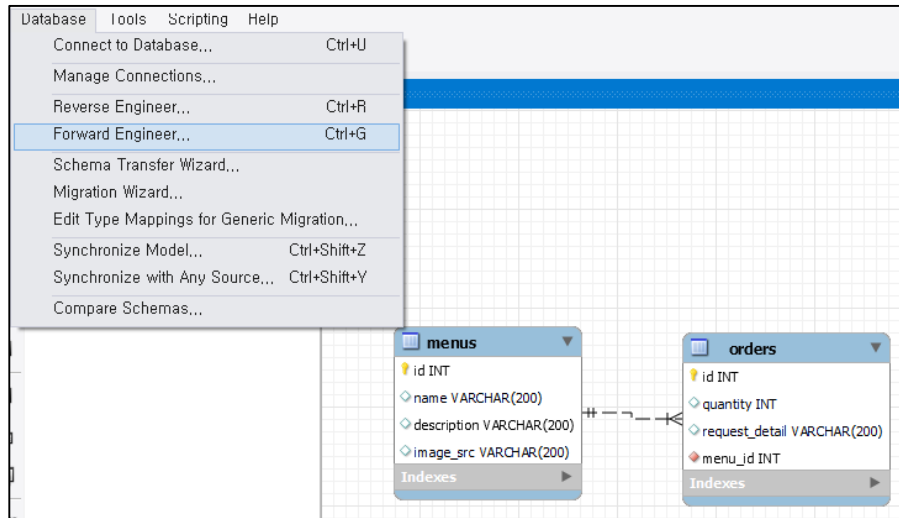
• N:M 관계

- 관계를 가진 양쪽에서 서로 1:N관계를 가지고 있는것
- 학원은 여러명의 수강생을 거느릴수 있다.
 - 수강생도 여러 학원을 다닐 수 있다.



만든 다이어그램을 기반으로 테이블 생성하기

- 왼쪽 상단 Database -> Forward Engineer
- 작성한 테이블을 기반으로 테이블을 생성



Set Options for Database to be Created

Tables

- ☐ Skip creation of FOREIGN KEYS
- ☐ Skip creation of FK Indexes as well
- ☐ Generate separate CREATE INDEX statements
- ☐ Generate INSERT statements for tables
- ☐ Disable FK checks for INSERTs

Other Objects

- ☐ Don't create view placeholder tables
- ☐ Do not create users. Only create privileges (GRANTS)

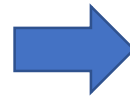
Code Generation

- ☐ DROP objects before each CREATE object
- ☐ Generate DROP SCHEMA
- ☐ Omit schema qualifier in object names
- ☐ Generate USE statements
- ☐ Add SHOW WARNINGS after every DDL statement
- ☒ Include model attached scripts

forward engineer시 에러 발생 대처법

- window 에 있는 workbench(8.x)의 mysql 버전과 aws의 mysql(5.x~) 버전이 다르게 되면 에러가 발생하기도 한다.
- INDEX 'fk~' VISIBLE 부분 제거후 실행

```
CREATE TABLE IF NOT EXISTS `order_system`.`orders` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `quantity` INT NULL,  
  `request_detail` VARCHAR(200) NULL,  
  `menus_id` INT NOT NULL,  
  PRIMARY KEY (`id`, `menus_id`),  
  INDEX `fk_orders_menus_idx` (`menus_id` ASC) VISIBLE,  
  CONSTRAINT `fk_orders_menus`  
    FOREIGN KEY (`menus_id`)  
    REFERENCES `order_system`.`menus` (`id`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION)  
ENGINE = InnoDB;
```



```
CREATE TABLE IF NOT EXISTS `order_system`.`orders` (  
  `id` INT NOT NULL AUTO_INCREMENT,  
  `quantity` INT NULL,  
  `request_detail` VARCHAR(200) NULL,  
  `menus_id` INT NOT NULL,  
  PRIMARY KEY (`id`, `menus_id`),  
  CONSTRAINT `fk_orders_menus`  
    FOREIGN KEY (`menus_id`)  
    REFERENCES `order_system`.`menus` (`id`)  
    ON DELETE NO ACTION  
    ON UPDATE NO ACTION)  
ENGINE = InnoDB;
```


왜 테이블을 둘로 나뉘었을까?

- 다음 상황을 가정해보자.
- 만약 100,000 개의 데이터를 저장한다면, 중복이 지나치게 많아 매우 비효율적
 - DB 용량 증가, 속도 저하
 - 비용 증가: Oracle 등의 유료 데이터베이스를 사용한다면, 훨씬 많은 비용 지불
 - 만약, menu_description 을 바꾸고 싶다면? 100,000 개의 데이터에서 찾아 전부 바꿔야한다

id	name	description	quantity	request_detail
1	카페라떼	라떼는.. 말이야	1	얼음 많이많이 주세요.
2	카페라떼	라떼는.. 말이야	3	시럽 넣지 말아주세요.
3	아메리카노	아메아메 좋아	1	
4	복숭아 아이스티	상큼한 복숭아!	2	

관계형 데이터베이스 (Relational Database)

- 여러 테이블의 "관계" 로 이루어진다.
- 핵심은 FK (Foreign Key, 외래키) 와 JOIN
- 중복 제거
 - 만약 100,000 개의 데이터를 저장하더라도,
 - DB 용량을 훨씬 효율적으로 사용
 - 만약, menu_description 을 바꾸고 싶다면? menus 테이블에서 "단 하나만" 변경하면 끝

menus			
id	name	description	image_src
1	카페라떼	라떼는.. 말이야	public/image1.jpg
2	카푸치노	언빌리 "버블"	public/image2.jpg
3	아메리카노	아메아메 좋아	public/image3.jpg
4	복숭아 아이스	상큼한 복숭아!	public/image4.jpg

orders			
id	quantity	request_detail	menus_id
1	1	얼음 많이많이 주세요.	1
2	3	시럽 넣지 말아주세요.	2
3	1	우유 많이 주세요.	2
4	2	복숭아 좋아요.	4

JOIN 문법을 사용하는 이유

- orders 에서, 주문마다 menus_id 보유
- 즉, id 만 알고 있다면 menus 에 접근해서
 - 메뉴 이름, 메뉴 설명, 메뉴 이미지에 대한 정보를 가져올 수 있음
 - 이를 위해 JOIN 문법 사용

menus			
id	name	description	image_src
1	카페라떼	라떼는.. 말이야	public/image1.jpg
2	카푸치노	언빌리 "버블"	public/image2.jpg
3	아메리카노	아메아메 좋아	public/image3.jpg
4	복숭아 아이스	상큼한 복숭아!	public/image4.jpg

orders			
id	quantity	request_detail	menus_id
1	1	얼음 많이많이 주세요.	1
2	3	시럽 넣지 말아주세요.	2
3	1	우유 많이 주세요.	2
4	2	복숭아 좋아요.	4



menus

- 메뉴 부터 먼저 작성해야한다.
- 메뉴 <-> 주문이 외래키로 엮여있기 때문에 주문을 작성하기 위해서는 메뉴 id가 필요하다.
 - 메뉴에 존재하지 않는 id를 주문의 menus_id에 넣는다면 에러가 발생할것
- worbench를 통해 작성하기

menus			
id	name	description	image_src
1	카페라떼	라떼는.. 말이야	public/image1.jpg
2	카푸치노	언빌리 "버블"	public/image2.jpg
3	아메리카노	아메아메 좋아	public/image3.jpg
4	복숭아 아이스	상큼한 복숭아!	public/image4.jpg

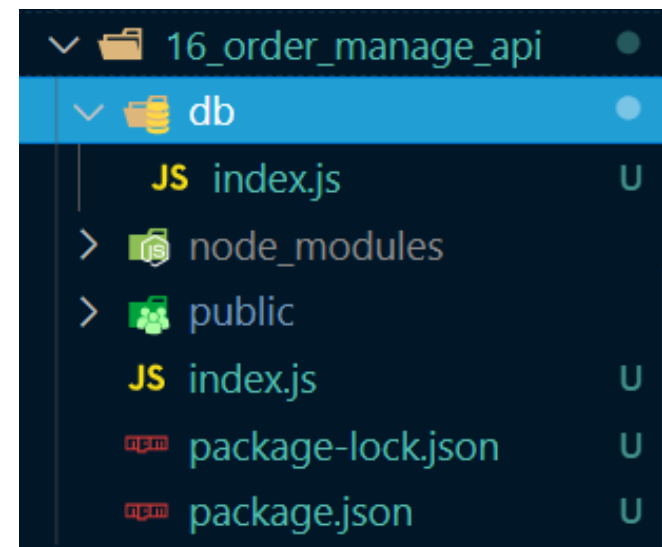
orders

- menus id값이 존재하는 경우만 menus_id 에 데이터를 기입할 수 있다.
- worbench를 통해 작성하기

orders			
id	quantity	request_detail	menus_id
1	1	얼음 많이많이 주세요.	1
2	3	시럽 넣지 말아주세요.	2
3	1	우유 많이 주세요.	2
4	2	복숭아 좋아요.	4

필요 라이브러리

- **express**
 - node.js 프레임워크
- **mysql2**
 - mysql 접속 라이브러리
- **cors**
 - cors 이슈 해결
- **morgan**
 - HTTP 요청에 대한 로그
- **multer**
 - 이미지 업로드를 위한 라이브러리
- **npm init으로 package.json 생성**
 - `npm i express mysql2 cors morgan multer`



```
"dependencies": {  
  "cors": "^2.8.5",  
  "express": "^4.18.1",  
  "morgan": "^1.10.0",  
  "multer": "^1.4.5-lts.1",  
  "mysql2": "^2.3.3"  
}
```

좋은 API 설계하기

- **유저 정보 조회**

- 유저 전체 정보 조회
 - GET /users
- 유저 등록
 - POST /users
- 특정 유저 조회
 - GET /users/:id
- 특정 유저 수정
 - PATCH /users/:id
- 특정 유저 삭제
 - DELETE /users/:id
- 대상에 대한 행동은 모두 HTTP REQUEST METHOD로 표시한다.
 - 행동과 리소스를 구별해서 설계하는것이 핵심이다.
 - 리소스 : 유저
 - 행동: 조회(GET), 등록(POST), 수정(PATCH), 삭제(DELETE)

필요한 기능을 생각해본다

- 메뉴 등록, 조회, 수정, 삭제

- menus

- GET /api/menus
 - 메뉴 전체 조회
 - GET /api/menus/:id
 - 메뉴 일부 조회
 - POST /api/menus
 - 메뉴 등록
 - PATCH /api/menus/:id
 - 메뉴 수정
 - POST /api/menus/:id/image
 - 메뉴 이미지 수정
 - DELETE /api/menus/:id
 - 메뉴 삭제

- 주문 하기, 주문 조회

- orders

- GET /api/orders
 - 주문 전체 조회
 - POST /api/orders
 - 주문하기

GET /api/menus

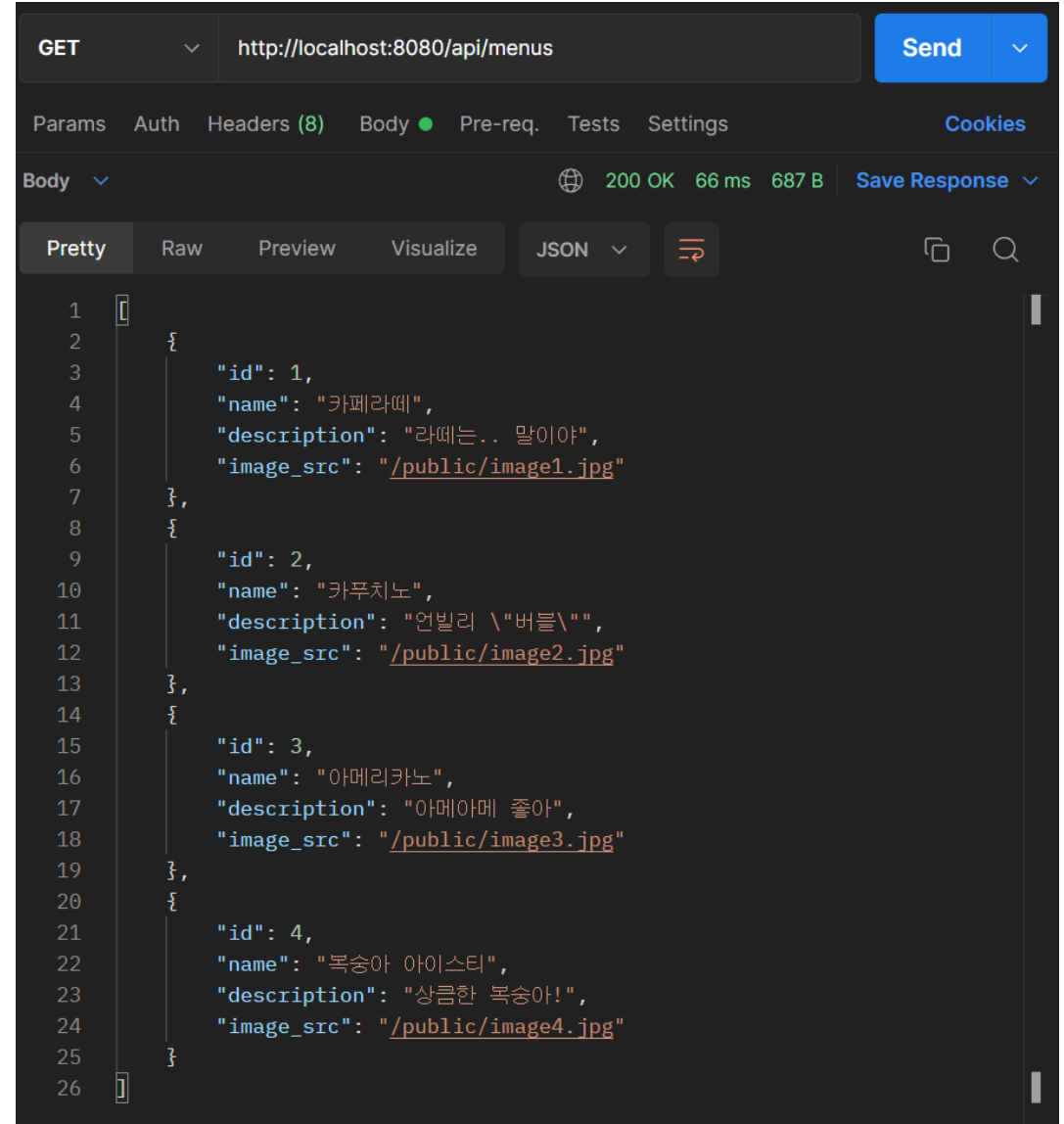
- 모든 메뉴 목록을 가져온다.
- 성공 시, 메뉴 목록 배열을 리턴하되, 각각의 프로퍼티는 다음과 같은 형태여야 한다.
 - [

```
{  
  "id": 1,  
  "name": "아이스 아메리카노",  
  "description": "여름엔 아아가 진리",  
  "image_src": "/public/ice-americano.jpg"  
}, ... ]
```
- 실패 시
 - { success : false, message: "전체 메뉴 목록 조회에 실패하였습니다." } 리턴

GET /api/menus 구현

```
app.get("/api/menus", async (req, res) => {  
  try {  
    const data = await pool.query("SELECT * FROM menus");  
    return res.json(data[0]);  
  } catch (error) {  
    return res.json({  
      success: false,  
      message: "전체 메뉴 목록 조회에 실패하였습니다."  
    });  
  }  
});
```

GET /api/menus 요청 결과



```
GET http://localhost:8080/api/menus 200 OK 66 ms 687 B Save Response
```

```
{
  "id": 1,
  "name": "카페라떼",
  "description": "라떼는.. 말이야",
  "image_src": "/public/image1.jpg"
},
{
  "id": 2,
  "name": "카푸치노",
  "description": "언빌리 \\"버블\\"",
  "image_src": "/public/image2.jpg"
},
{
  "id": 3,
  "name": "아메리카노",
  "description": "아메아메 좋아",
  "image_src": "/public/image3.jpg"
},
{
  "id": 4,
  "name": "복숭아 아이스티",
  "description": "상큼한 복숭아!",
  "image_src": "/public/image4.jpg"
}
```

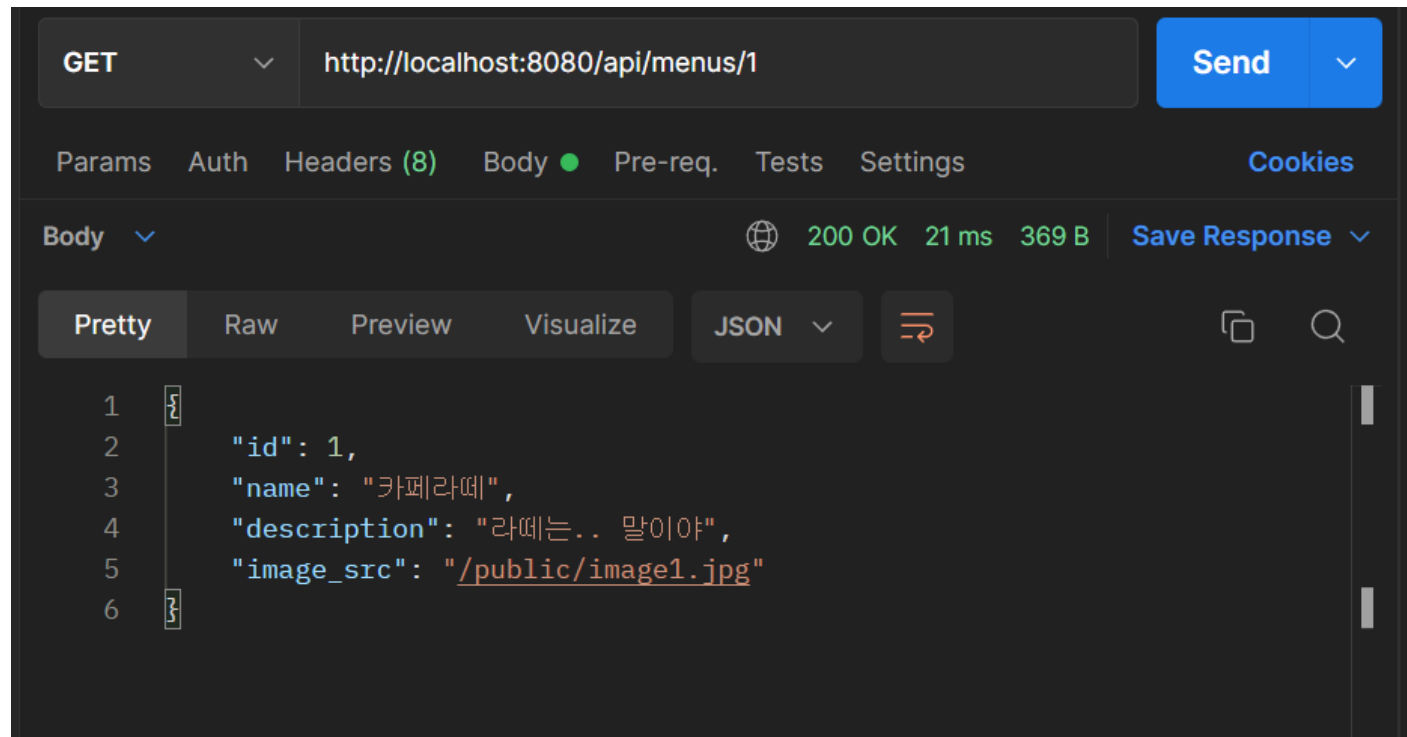
GET /api/menus/:id

- 메뉴 목록중 id에 해당하는 한가지를 가져온다.
- 성공 시, 메뉴 객체 리턴하되, 각각의 프로퍼티는 다음과 같은 형태여야 한다.
 - {
 "id": 1,
 "name": "아이스 아메리카노",
 "description": "여름엔 아아가 진리",
 "image_src": "/public/ice-americano.jpg"
}
- 실패 시
 - { success : false, message: " 메뉴 조회에 실패하였습니다." } 리턴

GET /api/menus/:id 구현

```
app.get("/api/menus/:id", async (req, res) => {  
  try {  
    const data = await pool.query("SELECT * FROM menus WHERE id = ?", [req.params.id]);  
    console.log(data[0]);  
    return res.json(data[0][0]);  
  } catch (error) {  
    console.log(error);  
    return res.json({  
      success: false, message: "메뉴 조회에 실패하였습니다."  
    });  
  }  
});
```

GET /api/menus/:id 요청 결과



POST /api/menus

- 메뉴를 추가한다.
- 클라이언트에서 서버로 보낼 JSON 은 다음과 같은 형태여야 한다.
 - {
 "name": "아이스 아메리카노",
 "description": "여름엔 아아가 진리",
 "file": "파일 첨부"
}
- 성공 시, 메뉴 목록 배열을 리턴하되, 각각의 프로퍼티는 다음과 같은 형태여야 한다.
 - {
 success:true,
 message: "메뉴 등록에 성공하셨습니다."
}
- 실패 시
 - {success : false, message: "메뉴 등록에 실패하였습니다." } 리턴

POST /api/menus 구현

- multer 라이브러리를 정의하여 이미지 업로드가 가능하게 해야한다.
- 이미지 업로드 후 해당 파일의 정보를 활용해 DB에 저장할 수 있어야 한다.
- 순서
 - 메뉴의 정보 + 파일 업로드
 - multer를 통해 파일 업로드 진행
 - 파일 업로드 완료후 해당 파일의 경로를 받아온다.
 - 받아온 경로 + 메뉴의 정보를 DB Mysql에 넣는다.

POST /api/menus

- multer 라이브러리 정의

- path.extname
 - 파일의 확장자만 가져온다
- fileNameExeptExt
 - 확장자를 제외한 파일 이름을 가져온다
- saveFileName
 - 확장자 제외 파일이름 + 현재 시간 + 확장자
 - ex) profile1661157882293.jpg

```
const path = require("path");
const multer = require("multer");
const upload = multer({
  storage: multer.diskStorage({
    destination: (req, file, done) => {
      done(null, "public/");
    },
    filename: (req, file, done) => {
      console.log(file);
      const ext = path.extname(file.originalname);
      console.log(ext);
      const fileNameExeptExt = path.basename(file.originalname, ext);
      console.log(fileNameExeptExt);
      const saveFileName = fileNameExeptExt + Date.now() + ext;
      done(null, saveFileName)
    },
  }),
  limits: { fileSize: 5 * 1024 * 1024 },
});
```

파일 이름에 Date.now() 를 추가하는 이유

- 확장자 제외 파일이름 + 현재 시간 + 확장자
- 예시
 - profile.jpg 라는 파일을 업로드했다.
 - 하지만 다른 사용자가 profile.jpg라는 파일을 또 업로드했다.
 - 기존의 profile.jpg가 덮어씌워진다.
 - 따라서 Date.now() 를 붙여서 현재 시 분 초의 정보를 같이 업로드하게 되면 사실상 파일 이름이 중복돼 덮어씌워질 일이 없다.

POST /api/menus 구현

- multer가 성공적으로 수행되면 req.file에 파일 정보가 담겨진다.
- pool.query("쿼리문 ?", [값1])
 - ? 개수만큼 배열에 대입이 가능

```
app.post('/api/menus', upload.single('file'), async(req, res) => {  
  try {  
    console.log(req.file);  
    console.log(req.file.path);  
    console.log(req.body);  
    const data = await pool.query(`INSERT INTO menus (name, description, image_src)  
    VALUES (?, ?, ?)`, [req.body.name, req.body.description, req.file.path]);  
  
    return res.json({  
      success: true, message: "메뉴 등록에 성공하였습니다."  
    });  
  } catch (error) {  
    console.log(error);  
    return res.json({  
      success: false, message: "메뉴 등록에 실패하였습니다."  
    });  
  }  
})
```

POST /api/menus 결과

POST http://localhost:8080/api/menus Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

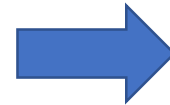
form-data

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	file	img3.jpg			
<input checked="" type="checkbox"/>	name	아이스 바닐라 라떼			
<input checked="" type="checkbox"/>	description	아 바 라!			

Body 200 OK 28 ms 335 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {  
2   "success": true,  
3   "message": "메뉴 등록에 성공하였습니다."  
4 }
```



Result Grid					Filter Rows:	Edit:	Export/Import:
	id	name	description	image_src			
	1	카페라떼	라떼는.. 말이야	public/image1.jpg			
	2	카푸치노	언빌리 "버블"	public/image2.jpg			
	3	아메리카노	아메아메 좋아	public/image3.jpg			
	4	복숭아 아이스티	상큼한 복숭아!	public/image4.jpg			
▶	10	아이스 바닐라 라떼	아 바 라!	publicWimg31661162914971.jpg			
*	NULL	NULL	NULL	NULL			

PATCH /api/menus/:id

- id를 받아와서 이미지를 제외한 메뉴를 수정한다.
 - 이미지는 POST 요청으로 보내야 한다.
- 클라이언트에서 서버로 보낼 JSON 은 다음과 같은 형태여야 한다.
 - {
 "name": "아이스 아메리카노",
 "description": "여름엔 아아가 진리",
}
- 성공 시, 메뉴 목록 배열을 리턴하되, 각각의 프로퍼티는 다음과 같은 형태여야 한다.
 - {
 success:true,
 message: "메뉴 정보 수정에 성공하셨습니다."
}
- 실패 시
 - {success : false, message: "메뉴 정보 수정에 실패하였습니다." } 리턴

PATCH /api/menus/:id 구현

```
app.patch('/api/menus/:id', async(req , res) => {  
  try {  
    console.log(req.params);  
  
    const data = await pool.query(`UPDATE menus SET name = ?, description = ? where id = ?`,  
    [req.body.name, req.body.description, req.params.id]);  
  
    return res.json({  
      success: true, message: "메뉴 정보 수정에 성공하였습니다."  
    });  
  } catch (error) {  
    console.log(error);  
    return res.json({  
      success: false, message: "메뉴 정보 수정에 실패하였습니다."  
    });  
  }  
})
```

PATCH /api/menus/:id 결과

The screenshot shows a REST client interface with the following details:

- Method:** PATCH
- URL:** http://localhost:8080/api/menus/10
- Body (JSON):**

```
{  1  {  2    "name": "바닐라 아이스크림",  3    "description": "부드러운 바닐라 아이스크림"  4  }
```
- Status:** 200 OK, 93 ms, 342 B
- Response (JSON):**

```
{  1  {  2    "success": true,  3    "message": "메뉴 정보 수정에 성공하였습니다."  4  }
```

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/api/menus/10
- Status:** 200 OK, 21 ms, 411 B
- Response (JSON):**

```
{  1  {  2    "id": 10,  3    "name": "바닐라 아이스크림",  4    "description": "부드러운 바닐라 아이스크림",  5    "image_src": "public\\img31661162914971.jpg"  6  }
```

POST /api/menus/:id/image

- id를 받아와서 이미지만 수정한다
 - 이미지는 POST 요청으로 보내야 하므로 POST로 처리한다.
- 클라이언트에서 서버로 보낼 JSON 은 다음과 같은 형태여야 한다.
 - {
 "file": "file 정보",
}
- 성공 시, 메뉴 목록 배열을 리턴하되, 각각의 프로퍼티는 다음과 같은 형태여야 한다.
 - {
 success:true,
 message: "메뉴 이미지 수정에 성공하셨습니다."
}
- 실패 시
 - {success : false, message: "메뉴 이미지 수정에 실패하였습니다." } 리턴

POST /api/menus/:id/image 구현

- 파일 업로드
- 업로드 후 src 업데이트

```
app.post('/api/menus/:id/image', upload.single('file'), async(req , res) => {  
  try {  
    const data = await pool.query(`UPDATE menus SET image_src= ? where id = ?`,  
      [req.file.path, req.params.id]);  
  
    return res.json({  
      success: true, message: "메뉴 이미지 수정에 성공하였습니다."  
    });  
  } catch (error) {  
    console.log(error);  
    return res.json({  
      success: false, message: "메뉴 이미지 수정에 실패하였습니다."  
    });  
  }  
})
```

POST /api/menus/:id/image 결과

GET http://localhost:8080/api/menus/10/ Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

form-data

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/>	file	img1.jpg			
	Key	Value	Description		

Body 200 OK 18 ms 411 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 10,
3   "name": "바닐라 아이스크림",
4   "description": "부드러운 바닐라 아이스크림",
5   "image_src": "public\\img11661163813635.jpg"
6 }
```

GET http://localhost:8080/api/menus/10/ Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

form-data

Body 200 OK 18 ms 411 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 10,
3   "name": "바닐라 아이스크림",
4   "description": "부드러운 바닐라 아이스크림",
5   "image_src": "public\\img11661163813635.jpg"
6 }
```

DELETE /api/menus/:id

- id에 해당되는 menus를 삭제한다.
- 요청시 보내는 JSON 데이터는 존재하지 않으며 :id로 삭제만 수행
- 성공 시, 메뉴 목록 배열을 리턴하되, 각각의 프로퍼티는 다음과 같은 형태여야 한다.
 - {
 success: true,
 message: "메뉴 삭제에 성공하셨습니다."
}
- 실패 시
 - {success : false, message: "메뉴 삭제에 실패하였습니다." } 리턴

DELETE /api/menus/:id 구현

```
app.delete('/api/menus/:id', async(req , res) => {  
  try {  
    const data = await pool.query(`DELETE FROM menus WHERE id = ?`,  
      [req.params.id]);  
  
    return res.json({  
      success: true, message: "메뉴 삭제에 성공하였습니다."  
    });  
  } catch (error) {  
    console.log(error);  
    return res.json({  
      success: false, message: "메뉴 삭제에 실패하였습니다."  
    });  
  }  
})
```

DELETE /api/menus/:id 결과

DELETE http://localhost:8080/api/menus/10

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body 200 OK 28 ms 345 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "success": true,
3   "message": "메뉴 이미지 삭제에 성공하였습니다."
4 }
```

GET http://localhost:8080/api/menus/10

Params Auth Headers (6) Body Pre-req. Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body 200 OK 17 ms 224 B Save Response

Pretty Raw Preview Visualize JSON

```
1
```

GET /api/orders

- 모든 주문 목록을 가져온다.
- 성공 시, 주문 목록 배열을 리턴하되, 각각의 프로퍼티는 다음과 같은 형태여야 한다.

- {
 "id": 1,
 "name": "아이스 아메리카노",
 "quantity": 1,
 "request_detail": "뜨겁게 만들어주세요"
 }, ...]

힌트: JOIN을 사용해 menus_id 에 해당하는 name을 가져와야한다.

- 실패 시
 - { success: false, message: "전체 주문 목록 조회에 실패하였습니다." } 리턴

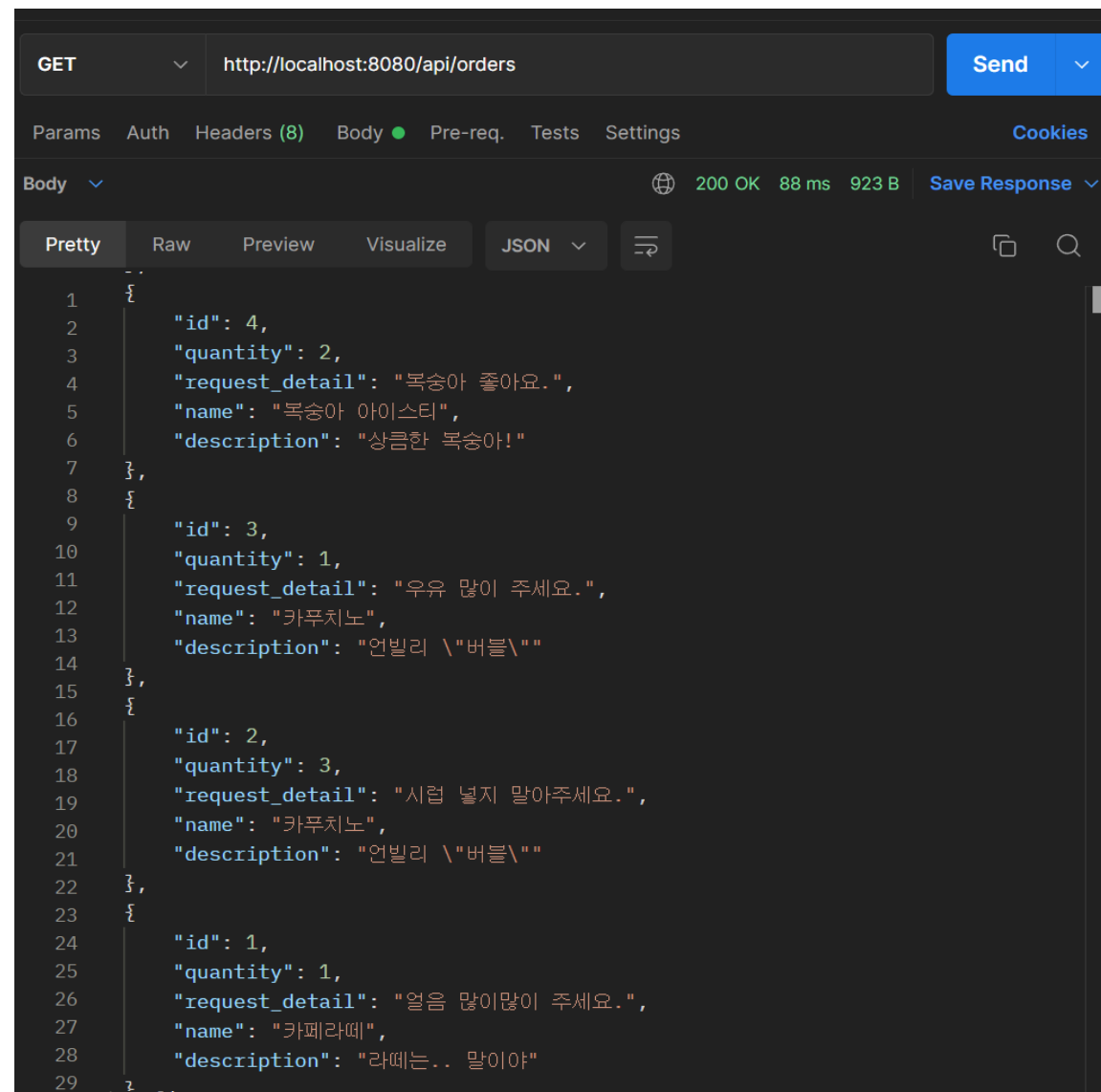
GET /api/orders 구현

- JOIN 활용
 - 주문수, 주문시 요청, 음료 이름, 음료 설명
 - a(orders)의 id인지 b(menus)의 id인지 명시
 - 명시 안하게 되는 경우 에러 발생
 - 컬럼의 이름이 서로 같으면 에러가 발생하기 때문에 id를 명시해줘야 한다.
 - id로 order by 정렬 진행

```
app.get("/api/orders", async (req, res) => {
  try {
    const data = await pool.query(`
      SELECT a.id, quantity, request_detail, name, description
      FROM orders as a
      INNER JOIN menus as b
      ON a.menu_id = b.id
      ORDER BY a.id DESC
    `);
    return res.json(data[0]);
  } catch (error) {
    console.log(error);
    return res.json({
      success: false,
      message: "전체 주문 목록 조회에 실패하였습니다."
    });
  }
});
```

GET /api/orders 결과

- Join을 통해 데이터가 잘 합쳐서 들어옴



```
GET http://localhost:8080/api/orders 200 OK 88 ms 923 B Save Response
```

```
{
  "id": 4,
  "quantity": 2,
  "request_detail": "복숭아 좋아요.",
  "name": "복숭아 아이스티",
  "description": "상큼한 복숭아!"
},
{
  "id": 3,
  "quantity": 1,
  "request_detail": "우유 많이 주세요.",
  "name": "카푸치노",
  "description": "언빌리 \\"버블\\"
},
{
  "id": 2,
  "quantity": 3,
  "request_detail": "시럽 넣지 말아주세요.",
  "name": "카푸치노",
  "description": "언빌리 \\"버블\\"
},
{
  "id": 1,
  "quantity": 1,
  "request_detail": "얼음 많이많이 주세요.",
  "name": "카페라떼",
  "description": "라떼는.. 말이야"
}
```


POST /api/orders

- 새로운 주문을 추가한다.
- 클라이언트에서 서버로 보낼 JSON 은 다음과 같은 형태여야 한다.
 - {
 "menus_id": 1
 "quantity": 1,
 "request_detail": "뜨겁게 주세요"
}
- 성공 시, 다음 형태의 JSON 을 리턴한다.
 - {
 success : true,
 id: 1 // 주문 번호이자 post시 생성된 orders의 id
}
- 실패 시
 - {success : false, message: "주문에 실패하였습니다." } 리턴

POST /api/orders 구현

```
app.post('/api/orders', async(req, res) => {  
  try {  
    const data = await pool.query(`INSERT INTO orders (quantity, request_detail, menus_id)  
    VALUES (?, ?, ?)`, [req.body.quantity, req.body.request_detail, req.body.menus_id]);  
    return res.json({  
      success: true, message: "주문에 성공하였습니다."  
    });  
  } catch (error) {  
    console.log(error);  
    return res.json({  
      success: false, message: "주문에 실패하였습니다."  
    });  
  }  
})
```

POST /api/orders 결과

POST `http://localhost:8080/api/orders` Send

Params Auth Headers (8) **Body** Pre-req. Tests Settings Cookies

raw JSON Beautify

```
1 {
2   "quantity": 3,
3   "request_detail": "안전하게 배달해 주세요",
4   "menus_id": 1
5 }
```

Body 200 OK 80 ms 328 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "success": true,
3   "message": "주문에 성공하였습니다."
4 }
```

GET `http://localhost:8080/api/orders` Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body 200 OK 75 ms 888 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "quantity": 3,
4     "request_detail": "안전하게 배달해 주세요",
5     "name": "카페라떼",
6     "description": "라떼는.. 말이야"
7   },
8   {
9     "quantity": 2,
10    "request_detail": "복숭아 좋아요.",
11    "name": "복숭아 아이스티",
12    "description": "상큼한 복숭아!"
13  },
14  {
15    "quantity": 1,
16    "request_detail": "우유 많이 주세요.",
17    "name": "카푸치노",
18    "description": "언빌리 \\"버블\\"
19  },
20 ]
```

orders 에 기능을 추가하기

- **GET /api/orders/:id**
 - 주문 내역 상세 조회 기능
- **PATCH /api/orders/:id**
 - 주문내역 수정 기능
- **DELETE /api/orders/:id**
 - 주문 내역 삭제 기능

GET /api/orders/:id

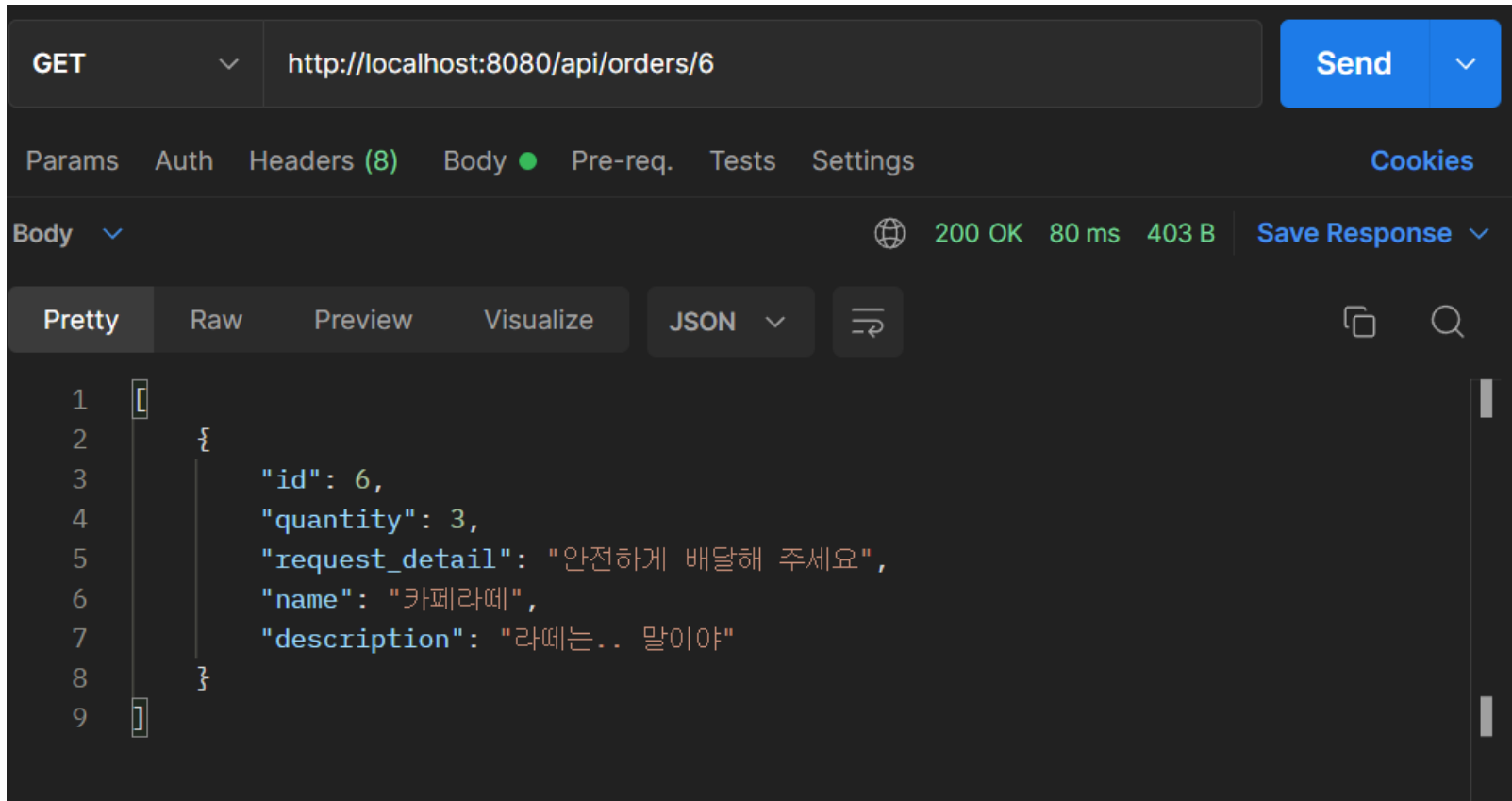
- 주문 내역을 가져온다.
- 성공 시, 주문 내역 객체를 리턴하되, 각각의 프로퍼티는 다음과 같은 형태여야 한다.

- {
 "id": 1,
 "name": "아이스 아메리카노",
 "quantity": 1,
 "request_detail": "뜨겁게 만들어주세요"
}

힌트: JOIN 사용해 menus_id 에 해당하는 name 가져와야한다.

- 실패 시
 - { success: false, message: "주문 내역 조회에 실패하였습니다." } 리턴

GET /api/orders/:id 결과



PATCH /api/orders/:id

- 기존 주문 내역을 수정한다.
- 클라이언트에서 서버로 보낼 JSON 은 다음과 같은 형태여야 한다.
 - {
 "menus_id": 1
 "quantity": 1,
 "request_detail": "뜨겁게 주세요"
}
- 성공 시, 다음 형태의 JSON 을 리턴한다.
 - {
 success : true,
 message: "주문 수정에 성공하셨습니다."
}
- 실패 시
 - {success : false, message: "주문 수정에 실패하였습니다." } 리턴

PATCH /api/orders/:id 결과

PATCH `http://localhost:8080/api/orders/6` Send

Params Auth Headers (8) **Body** Pre-req. Tests Settings Cookies Beautify

raw JSON

```
1 {
2   "quantity": 3,
3   "request_detail": "종이 빨대는 빼고 주세요.",
4   "menus_id": 2
5 }
```

Body 200 OK 140 ms 342 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "success": true,
3   "message": "메뉴 정보 수정에 성공하였습니다."
4 }
```



GET `http://localhost:8080/api/orders/6` Send

Params Auth Headers (8) **Body** Pre-req. Tests Settings Cookies Beautify

raw JSON

```
1 {
2   "quantity": 3,
3   "request_detail": "종이 빨대는 빼고 주세요.",
4   "menus_id": 2
5 }
```

Body 200 OK 20 ms 404 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 6,
3   "quantity": 3,
4   "request_detail": "종이 빨대는 빼고 주세요.",
5   "name": "카푸치노",
6   "description": "언빌리 버블"
7 }
8
9
```


DELETE /api/orders/:id

- 기존 주문 내역을 삭제한다.
- 성공 시, 다음 형태의 JSON 을 리턴한다.
 - {
 success : true,
 message: "주문 삭제에 성공하셨습니다."
}
- 실패 시
 - {success : false, message: "주문 삭제에 실패하였습니다." } 리턴

DELETE /api/orders/:id 결과

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:8080/api/orders/6`
- Method:** `DELETE`
- Status:** `200 OK` (76 ms, 342 B)
- Response Body (JSON):**

```
{  "success": true,  "message": "주문 내역 삭제에 성공하였습니다."}
```

The interface also includes tabs for Params, Auth, Headers (8), Body, Pre-req., Tests, Settings, and Cookies. The Body tab is currently selected, showing the response in JSON format.

API 서버 배포

MobaXterm 접속

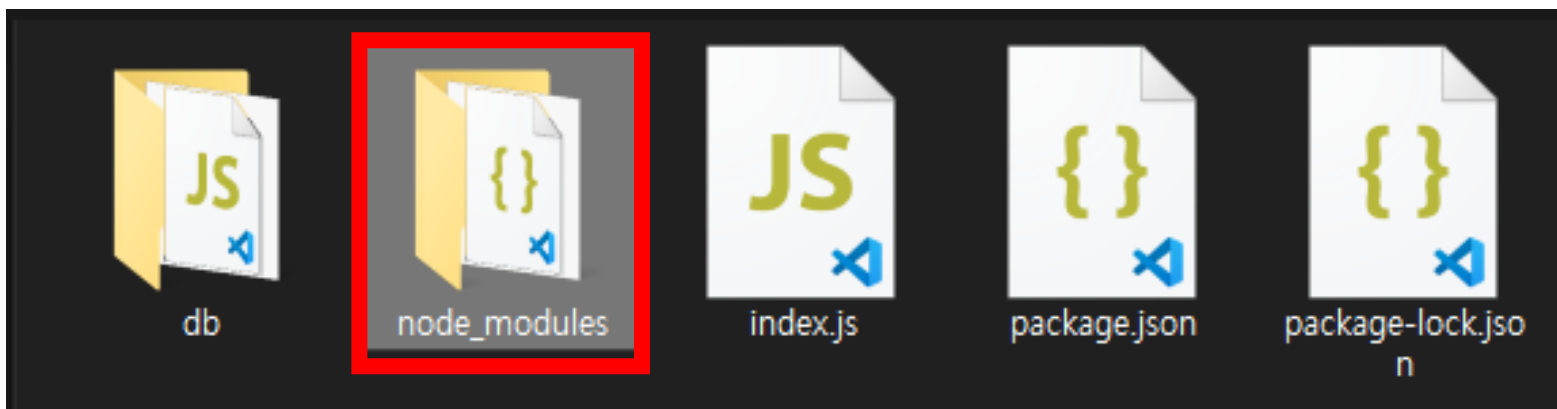
- AWS 인스턴스에 Node.js LTS 16 버전 설치하기

- `curl -fsSL https://deb.nodesource.com/setup_lts.x | sudo -E bash -`
- `sudo apt update`
- `sudo apt install -y nodejs`
- 버전체크
 - `node -v`
 - `npm -v`

```
ubuntu@ip-172-31-40-222:~$ node -v
v16.15.1
ubuntu@ip-172-31-40-222:~$ npm -v
8.11.0
```

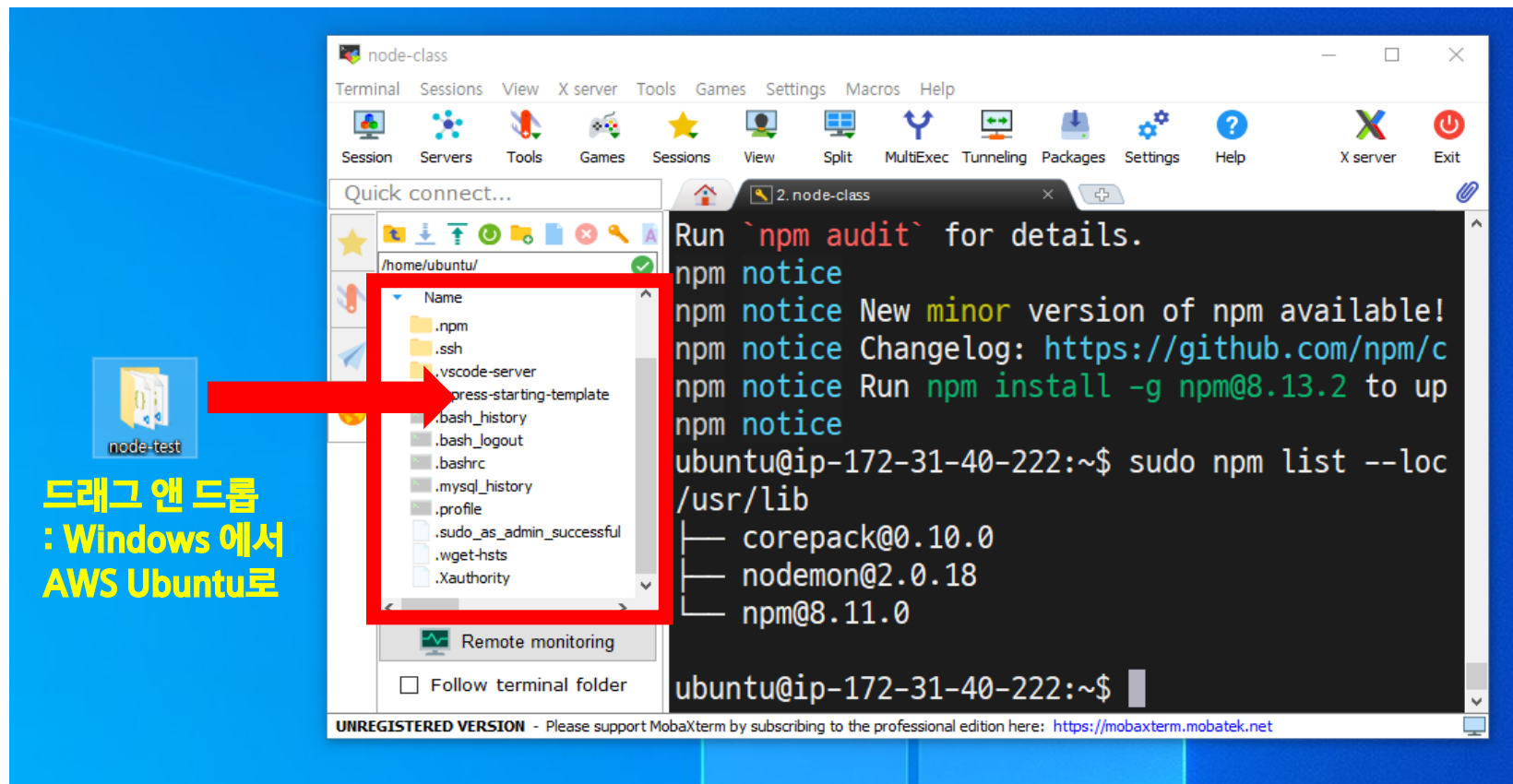
node_modules 제외하기

- node_modules는 용량을 상당히 차지한다.
- 프로젝트 디렉터리에서 node_modules를 삭제하거나 해당 부분을 빼고 업로드
- 업로드 된 서버에서 해당 node_modules를 package.json을 통해 재 설치
- github를 통해 협업시 .gitignore 파일에 node_modules를 추가
 - github에 node_modules가 올라가지 않도록 한다.



API 서버 배포

- 작성한 node 폴더를 드래그&드롭을 통해 AWS ubuntu에 넣는다.



프로젝트 디렉터리로 이동

```
$ cd ~/프로젝트이름
```

```
$ ls -al
```

필요한 패키지 설치

- `sudo npm i`

서버 구동

- `sudo node index.js`
- `nodemon`을 사용하지 않는 이유
 - 운영 환경은 자동 재시작이 되면 X

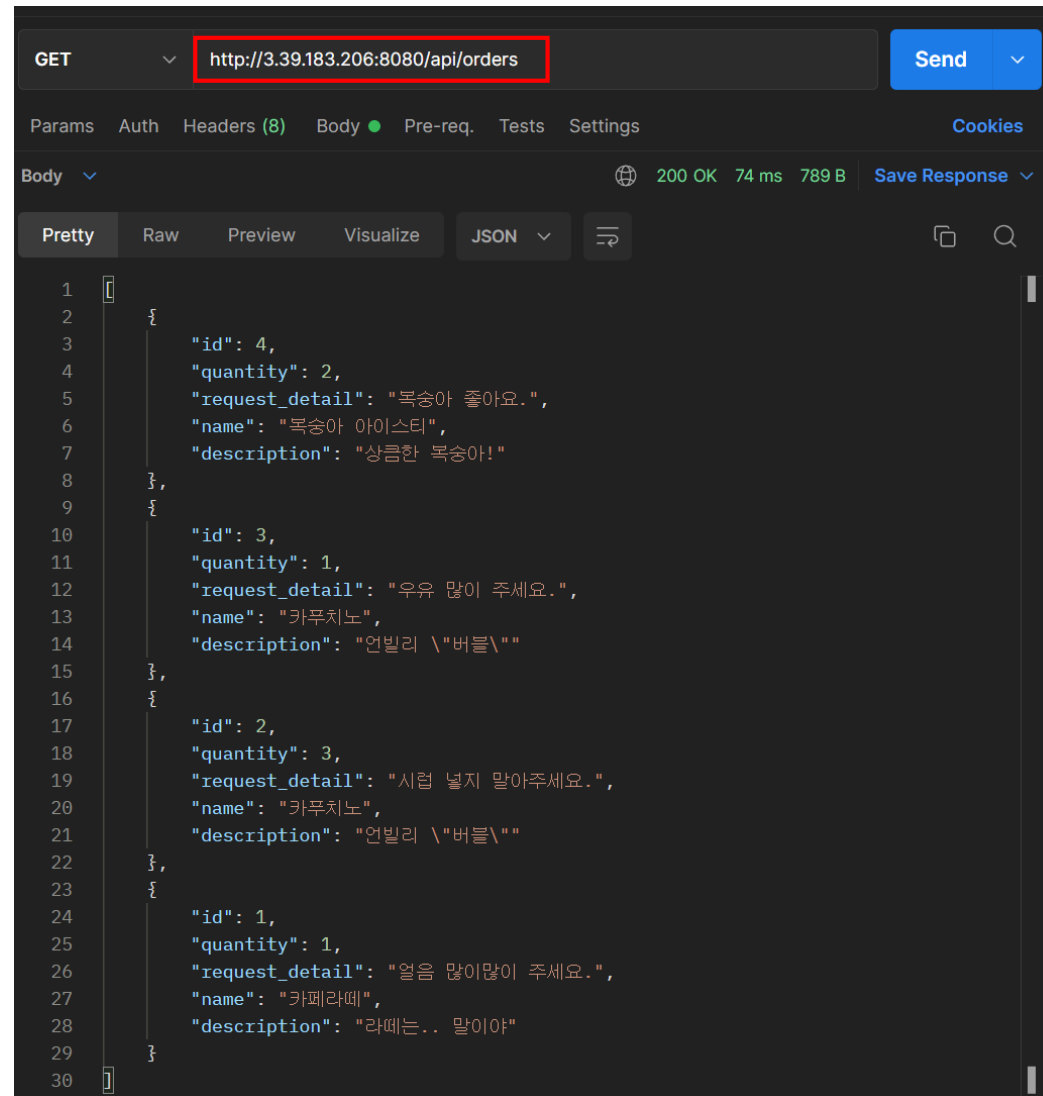
```
ubuntu@ip-172-31-30-8:~/16_order_manage_api$ sudo npm i
changed 96 packages, and audited 97 packages in 3s

7 packages are looking for funding
  run `npm fund` for details

ubuntu@ip-172-31-30-8:~/16_order_manage_api$ node index
this server listening on 8080
```

POSTMAN 요청 주소 변경

- AWS 주소:8080에 요청을 보내보기
- 이전에 지정한 도메인 주소
 - https://내도메인.한국에 접속및 로그인
 - 해당 도메인:8080/api/orders 에 요청보내기



서버 상시 유지하기

- 현재 MobaXterm을 접속 종료하거나 Ctrl + C 를 눌러 서버를 나가게 되면 더 이상 요청이 동작하지 않는다.
- 백그라운드에서 서버를 실행시켜 주는 작업이 필요
- `sudo nohup node index.js &`
 - 해당 명령어를 실행하면 node가 터미널이 닫히더라도 백그라운드에서 실행이 가능하다.

```
ubuntu@ip-172-31-30-8:~/16_order_manage_api$ sudo nohup node index.js &  
[1] 30981  
ubuntu@ip-172-31-30-8:~/16_order_manage_api$ nohup: ignoring input and appending output to 'nohup.out'
```

백그라운드 된 서버 종료하는 법

- **ps -ef |grep node 명령어 실행**
 - 빨간색으로 가리킨 곳이 PID 파트
 - `sudo kill -9 PID`
 - 실행하면 백그라운드로 서비스되고있는 node를 종료 가능하다.
 - ex) `sudo kill -9 30991`
 - 30991 PID를 가지고있는 서비스 종료

```
ubuntu@ip-172-31-30-8:~/16_order_manage_api$  
ubuntu@ip-172-31-30-8:~/16_order_manage_api$  
ubuntu@ip-172-31-30-8:~/16_order_manage_api$ ps -ef |grep node  
root      30981 29429  0 11:42 pts/0    00:00:00 sudo nohup node index.js  
root      30982 30981  0 11:42 pts/0    00:00:00 node index.js  
ubuntu    30991 29429  0 11:43 pts/0    00:00:00 grep --color=auto node  
ubuntu@ip-172-31-30-8:~/16_order_manage_api$ sudo kill -9 30928 30981
```

내일 방송에서 만나요!

삼성 청년 SW 아카데미