주간 졸업 프로젝트 수행 내용

전임자 김민수가 짠 코드를 전체적으로 파악을 하였다.

코드가 너무 지저분하고, 설명도 안 되어있어서 전체적인 파악 하느라 시간이 너무 오래 걸렸다.

코드 파악을 한 후에, Frontend 부분에서 footer과 header 중복되는 부분을 모듈로 묶은 후에, 각 코드마다 footer과 header을 연결 시켰다.

그 후에는 코드 파악 중에 전임자가 DB를 직접 SQL문으로 날리는 식으로 구현해 놓고 가서, 보안상의 문제도 있고, 코드 자체도 지저분해 보이고, 보기 안 좋아서 고민을 해 보다가, Node.js 교과서에서 본 DB를 객체로 관리하는 라이브러리를 찾았다.

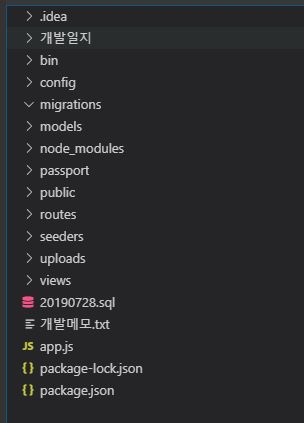
그게 바로 Sequelize 라는 라이브러리인데, Sequelize는 ORM(Object-Relational Mapping)으로 분류되며, ORM은 자바스크립트 객체와 데이터베이스 릴레이션을 매핑해주는 도구이다.



Sequelize Image

개인적으로는 SQL문을 직접 써서 구현하면 좋긴 하지만, 관계를 맺어서 데이터를 찾을 때 퀴리문이 지저분해지므로, Sequelize로 처리하면 코드도 매우 깔끔해지고, 가독성에서도 좋은 듯 하다.

다음은 프로젝트 구조이다.



Project 구조

Views는 화면을 보여주는 FrontEnd 부분이고, DB 모델링은 models에서 실시하였다.

1. models index.js

|  |
| --- |
| const path = require('path');  const Sequelize = require('sequelize');  const env = process.env.NODE\_ENV || 'development';  const config = require(path.join(\_\_dirname, '..', 'config', 'config.json'))[env];  const db = {};  const sequelize = new Sequelize(config.database, config.username, config.password, config);  db.sequelize = sequelize;  db.Sequelize = Sequelize;  db.Faulty = require('./faulty')(sequelize,Sequelize);  db.Release = require('./release')(sequelize,Sequelize);  db.Sales = require('./sales')(sequelize,Sequelize);  db.Stationery = require('./stationery')(sequelize,Sequelize);  db.Stock = require('./stock')(sequelize,Sequelize);  db.Training\_result = require('./training\_result')(sequelize,Sequelize);  db.User = require('./user')(sequelize,Sequelize);  db.User.hasMany(db.Sales, {foreignKey : 'User\_NO', sourceKey : 'id'});  db.Sales.belongsTo(db.User, {foreignKey : 'User\_NO', targetKey : 'id'});  db.User.hasMany(db.Faulty, {foreignKey : 'User\_NO', sourceKey : 'id'});  db.Faulty.belongsTo(db.User, {foreignKey : 'User\_NO', targetKey : 'id'});  db.User.hasMany(db.Release, {foreignKey : 'User\_NO', sourceKey : 'id'});  db.Release.belongsTo(db.User, {foreignKey : 'User\_NO', targetKey : 'id'});  db.Stationery.hasMany(db.Training\_result, {foreignKey : 'Stationery\_NO', sourceKey : 'id'});  db.Training\_result.belongsTo(db.Stationery, {foreignKey : 'Stationery\_NO', targetKey : 'id'});  db.Stationery.hasMany(db.Faulty, {foreignKey : 'Stationery\_NO', sourceKey : 'id'});  db.Faulty.belongsTo(db.Stationery, {foreignKey : 'Stationery\_NO', targetKey : 'id'});  db.Stationery.hasMany(db.Sales, {foreignKey : 'Stationery\_NO', sourceKey : 'id'});  db.Sales.belongsTo(db.Stationery, {foreignKey : 'Stationery\_NO', targetKey : 'id'})  db.Stationery.hasMany(db.Stock, {foreignKey : 'Stationery\_NO', sourceKey : 'id'});  db.Stock.belongsTo(db.Stationery, {foreignKey : 'Stationery\_NO', targetKey : 'id'});  db.Stationery.hasMany(db.Release, {foreignKey : 'Stationery\_NO', sourceKey : 'id'});  db.Release.belongsTo(db.Stationery, {foreignKey : 'Stationery\_NO', targetKey : 'id'});  module.exports = db; |

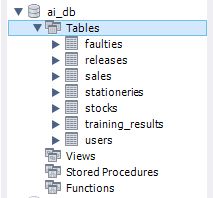
다음과 같이 자바스크립트로 생성된 테이블을 Join을 할 수 있다.

2. models user.js

|  |
| --- |
| module.exports = (sequelize, DataTypes) => {  return sequelize.define('user', {  Name : {  type : DataTypes.STRING(100),  allowNull : false,  },  Email : {  type : DataTypes.STRING(15),  allowNull : true,  },  PW : {  type : DataTypes.STRING(20),  allowNull : false,  },  HP : {  type : DataTypes.STRING(10),  allowNull : true,  },  Position : {  type : DataTypes.STRING(50),  allowNull : false,  },  }, {  timestamps : false,  });  }; |

다음과 같이 user 테이블이 생성 되었다.

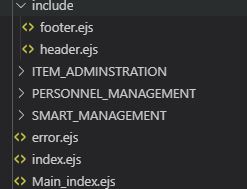
그 외 테이블 생성 코드는 다음 위에 같으니 생략



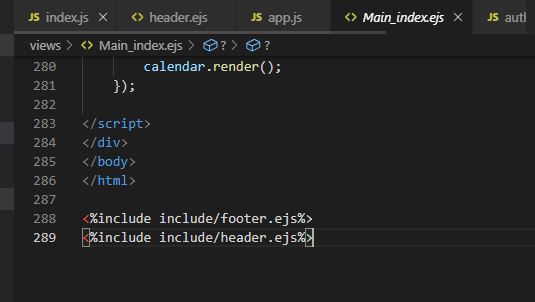
생성된 DB 결과

DB부분 생성은 쉽게 되었다.

FrontEnd에서 Footer과 header 부분은 다음과 같이 폴더를 생성하여 include를 시켰다.



Footer과 Header 부분 ejs 파일 생성하여 객체로 묶음



Footer.ejs와 header.ejs Include 부분

더 나아가 로그인까지 구현하고 싶었지만, 로그인 과정에서 오류가 너무 많아서 로그인 과정은 다음주에 완료 목표를 잡기로 하겠다.

다음주 수행 과제

1. passport 라이브러리 활용하여 로그인, 회원가입 구현

2. 홈페이지 기능 전반적으로 재구성