(1)	Express - Type Script

Ⓐ 프로젝트 초기화

```
npm init -y
npx tsc --init ( npm install –g typescript )
```

① package.json

```
"scripts": {
  "start": "node dist/index.js",
  "dev": "nodemon src/index.ts",
  "build": "rm -rf dist/ && tsc -p ."
}
```

② tsconfig.json

```
{
  "compilerOptions": {
    "target": "es6",
    "strict": true,
    "skipLibCheck": true,
    "sourceMap": true,
    "rootDir": "./src",
    "outDir": "./dist",
    "module": "commonjs",
    "moduleResolution": "node",
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true,
```

```
  },
  "include": ["src/**/*.ts"],
}
```

Ⓑ Express, TypeScript 기본 설치

```
npm install --save express
npm install --save-dev nodemon typescript ts-node @types/node @types/express
```

① ts-node (ts-node-dev)

ⓐ node 상에서, 독립적으로 .ts 파일을 실행한다.

ⓑ node 상에서 TypeScript Compiler를 통하지 않고도, 직접 TypeScript를 실행시키는 역할을 해줌

② @types/??

모듈에 대한 타입을 지원(npm의 DefinitelyTyped 패키지에서 제공하는 커뮤니티 타입 정의), 실제 서버를 구동할때 사용되지 않는다.

ⓐ express - @types/express

ⓑ dotenv - @types/dotenv

ⓒ morgan - @types/morgan

ⓓ passport - @types/passport

ⓔ express-session - @types/express-session

ⓕ cookie-parser - @types/cookie-parser

ⓖⓗⓘⓙⓚⓛⓜⓝⓞⓟⓠⓡⓢⓣⓤⓥⓦⓧⓨⓩ

```
npm install —save bcrypt cookie-parser cors dotenv express-session morgan mysql2 passport sequelize
```

```
npm install —save-dev sequelize-cli @types/bcrypt
```

```
@types/cookie-parser  @types/cors  @types/dotenv
@types/express-session  @types/morgan
@types/passport  @types/sequelize
```

③

Ⓒ Git

```
git init
  git config --global user.name "ipfs-protocol"
  git config --global user.email "ipfs-protocol@gamil.com"
  git status
  git config —list
git add .
git commit -m "commit"
git branch –M master
git remote add origin https://github.com/ipfs-protocol/exp
ress.git
git push -u origin master
```

Ⓓ DefinitelyTyped 패키지

```
bcrypt cors morgan sequelize mysql2
```

```
@types/bcrypt @types/cors
    @types/sequelize
```

==========

(1)(2)(3)(4)(5)(6)(7)(8)(9)(10)(11)

⑿ ⒀ ⒁ ⒂ ⒃ ⒄ ⒅ ⒆ ⒇ ㈠

㈡㈢㈣㈥㈦㈨㈩◎

ⒶⒷⒸⒹⒺⒻⒼⒽⒾⒿⓀⓁⓂⓃⓄⓅⓆⓇⓈⓉⓊⓋⓌⓍⓎⓏ

①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚

ⓐⓑⓒⓓⓔⓕⓖⓗⓘⓙⓚⓛⓜⓝⓞⓟⓠⓡⓢⓣⓤⓥⓦⓧⓨⓩ

㊀㊁㊂㊃㊄㊅㊆㊇㊈㊉

• ○ ✪ ↳ ▪ ▸ ↱ ↳ ⇨ ❝ ❞ ❛ ❜ ⇓ ▪ → ⇦

==========

(2)	MySQL

Ⓐ Data Type

① 숫자 데이터형

ⓐ 정수 데이터형

```
INT(n)
```

㉠ TINYINT(1) : -128 ~ 127, 0 ~ 255

㉡ SMALLINT(2) : -32768 ~ 32767, 0 ~ 65535

㉢ MEDIUMINT(3) : -8388608 ~ 8388607, 0 ~ 16777215

㉣ INT(4) : -2147483648 ~ 2147483647, 0 ~ 4294967295

㉤ BIGINT(8) : -263(약 –922경) ~ 263-1(약 922경)

ⓑ 실수 데이터형

```
FLOAT(n,m)
```

㉠ FLOAT(4) : 소수점을 포함하여 값을 저장

② 문자 데이터형

ⓐ CHAR

```
CHAR(n)
```

㉠ 1~255 바이트까지 고정길이 문자열을 저장

㉡ 정의된 공간보다 입력 데이터가 짧으면 나머지 공간은 공백(SPACE)으로 채워지고, 데이터가 길면 맞게 잘린 데이터가 입력된다(길어도 에러 발생하지 않는다).

ⓑ VARCHAR

```
VARCHAR(n)
```

㉠ 최대 255 바이트까지 저장하는 가변 길이 문자열 저장

㉡ 정의된 공간보다 입력 데이터가 길면 에러값을 리턴한다.

ⓒ BLOB, TEXT

㉠ 65,535 이상의 거대 텍스트 데이터를 저장

㉡ BLOB는 검색시 대소문자를 구별하나 TEXT는 대소문자의 구분이 없이 검색한다.

㉢ MEDIUMTEXT(MEDIUMBLOB) : 16,777,215 문자열

㉣ LONTTEXT(LONGBLOB) : 4,294,967,295(4G) 문자열

③ 날짜 데이터형

ⓐ DATE

ⓑ DATETIME

ⓒ TIMESTAMP

```
TIMESTAMP(m)
```

m에 따라 다양한 형태의 날짜 저장

ⓓ TIME

ⓔ YEAR

ⓕ DATE

④ 바이너리 데이터형

ⓐ RAW 데이터형

이진형 테이터를 255 바이트까지 수용할 수 있으나 저장공간의 제한점 때문에 많이 사용하지 않는다.

ⓑ LONG RAW 데이터형 : 이진형 데이터를 2G까지 저장

ⓒ BLOB 데이터형 : 이진형 데이터를 4G까지 저장

ⓓⓔⓕ

⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚

ⓐⓑⓒⓓⓔⓕⓖⓗⓘⓙⓚⓛⓜⓝ

㈠㈡㈢㈣㈤㈥㈦㈧㈨㈩

Ⓑ 사용자 생성

```
// mysql에 있는 host와 user 정보를 가져온다.
SELECT host, user FROM mysql.user
// 사용자와 비밀번호를 설정한다.
CREATE User '사용자이름'@'%' identified by '비밀번호'
// 사용자의 권한을 설정한다.
GRANT ALL PRIVILEGES ON DB명.* TO '사용자명'@'%'
// 설정한 접근권한을 저장
FLUSH PRIVILEGES
```

ⒸⒹⒺⒻⒼⒽⒾⒿⓀⓁⓂ

①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚

ⓐⓑⓒⓓⓔⓕⓖⓗⓘⓙⓚⓛⓜⓝ

Ⓐ user

```
CREATE TABLE nodejs.users(
  id INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(20) NOT NULL,
  age INT UNSIGNED NOT NULL,
  gender TINYINT NOT NULL,
  comment TEXT NULL,
  created_at DATETIME NOT NULL DEFAULT now(),
  PRIMARY KEY(id),
```

```
  UNIQUE INDEX name_UNIQUE (name ASC)
)
COMMENT='사용자 정보'
DEFAULT CHARACTER SET = utf8
ENGINE = InnoDB;
```

Ⓑ comment

```
CREATE TABLE nodejs.comment(
  id INT NOT NULL AUTO_INCREMENT,
  commenter INT NOT NULL,
  comment VARCHAR(100) NOT NULL,
  created_at DATETIME NOT NULL DEFAULT now(),
  PRIMARY KEY(id),
  INDEX commenter_idx (commenter ASC),
  CONSTRAINT commenter
  FOREIGN KEY (commenter)
  REFERENCES nodejs.users (id)
  ON DELETE CASCADE
  ON UPDATE CASCADE
)
COMMENT='댓글'
DEFAULT CHARSET = utf8mb4
ENGINE = InnoDB;
```

ⒸⒹⒺⒻⒼ

(3)	Sequelize
	MySQL 작업을 쉽게 할 수 있도록 도와 주는 라이브러리

Ⓐ Sequelize

① ORM(Object-relational Mapping)로 분류된다.
② 자바스크립트 구문을 자동으로 SQL로 바꾸어준다.

Ⓑ 설치

```
npm install sequelize sequelize-cli mysql2
```

① sequelize-cli : 시퀄라이즈 명령어를 실행하기 위한 패키지
② mysql2 : MySQL과 시퀄라이즈를 이어주는 드라이버

Ⓒ Sequelize 초기화

```
npx sequelize init
```

① config, models, migrations, seeders 폴더 생성
② models 폴더에 index.js가 생성, 수정

```
// MySQL, node, sequelize 연결 코드
const Sequelize = require('sequelize');

const env = process.env.NODE_ENV || 'development';
const config = require('../config/config')[env];
const db = {};
```

```
const sequelize = new Sequelize(config.database, config.username, config.password, config);

db.sequelize = sequelize;

module.export = db;
```

③ config/config.json

```
{
  "development" {
    "username": "root",
    "password": "girl",
    "database": "nodejs",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  ''''''
}
```

ⓓ MySQL – Sequelize 자료형

MySQL	Sequelize
VARCHAR(100)	STRING(100)
INT	INTEGER
TINYINT	BOOLEAN

DATETIME	DATE
INT UNSIGNED	INTERGER.UNJSIGNED
NOT NULL	allowNull: false
UNIQUE	unique: true
DEFAULT now()	defaultValue: Sequelize.NOW

ⓔ 모델(Table) 정의하기

MySQL의 테이블은 시퀄라이즈의 모델과 대응합니다. 시퀄라이즈는 모델과 MySQL의 테이블을 연결해주는 역할을 합니다.

① MySQL/user.sql - models/user.js

```
CREATE TABLE nodejs.users(
  id INT NOT NULL AUTO_INCREMENT,
  name VARCHAR(20) NOT NULL,
  age INT UNSIGNED NOT NULL,
  married TINYINT NOT NULL,
  comment TEXT NULL,
  created_at DATETIME NOT NULL DEFAULT now(),
  PRIMARY KEY(id),
  UNIQUE INDEX name_UNIQUE (name ASC)
)
COMMENT='사용자 정보'
DEFAULT CHARACTER SET = utf8
ENGINE = InnoDB;
```

```
const Sequelize = require('sequelize');
module.exports = class User extends Sequelize.Model {
  // User : 모델이름(테이블 이름)
  static init(sequelize) {
    // init( {데이터 설정}, {모델 설정} )
    return super.init({
      name: {
        type: Sequelize.STRING(20),
        allowNull: false,
        unique: true,
      },
      age: {
        type: Sequelize.INTEGER.UNSIGNED,
        allowNull: false,
      },
      married: {
        type: Sequelize.BOOLEAN,
        allowNull: false,
      },
      comment: {
        type: Sequelize.TEXT,
        allowNull: true,
      },
      created_at: {
        type: Sequelize.DATE,
        allowNull: false,
        defaultValue: Sequelize.NOW,
```

```
      },
    }, {
      sequelize,
      timestamps: false,
      underscored: false,
      modelName: 'User',
      tableName: 'users',
      paranoid: false,
      charset: 'utf8',
      collate: 'utf8_general_ci',
    });
  }

  static associate(db) {
    db.User.hasMany(db.Comment, { foreignKey:
    'commenter', sourceKey: 'id' });
  }
};
```

ⓐ

ⓑⓒⓓⓔⓕⓖⓗⓘⓙⓚⓛⓜⓝⓞⓟⓠⓡⓢⓣⓤⓥⓦⓧⓨⓩ

② models/user.js

```

```

ⓐ 시퀄라이즈는 id는 자동으로 생성된다.

ⓑⓒⓓⓔⓕⓖⓗⓘⓙⓚⓛⓜⓝⓞⓟⓠⓡⓢⓣⓤⓥⓦⓧⓨⓩ

```
const Sequelize = require('sequelize');
module.exports = class User extends Sequelize.Model {
  // User : 모델이름(테이블 이름)
  static init(sequelize) {
    // init( {데이터 설정}, {모델 설정} )
    return super.init({
      name: {
        type: Sequelize.STRING(20),
        allowNull: false,
        unique: true,
      },
      age: {
        type: Sequelize.INTEGER.UNSIGNED,
        allowNull: false,
      },
      married: {
        type: Sequelize.BOOLEAN,
        allowNull: false,
      },
      comment: {
        type: Sequelize.TEXT,
        allowNull: true,
      },
      created_at: {
        type: Sequelize.DATE,
        allowNull: false,
        defaultValue: Sequelize.NOW,
```

```
      },
    }, {
      sequelize,
      timestamps: false,
      underscored: false,
      modelName: 'User',
      tableName: 'users',
      paranoid: false,
      charset: 'utf8',
      collate: 'utf8_general_ci',
    });
  }

  static associate(db) {
    db.User.hasMany(db.Comment, { foreignKey:
    'commenter', sourceKey: 'id' });
  }
};
```

③ MySQL/comment.sql

```
CREATE TABLE nodejs.comment(
  id INT NOT NULL AUTO_INCREMENT,
  commenter INT NOT NULL,
  comment VARCHAR(100) NOT NULL,
  created_at DATETIME NOT NULL DEFAULT now(),
  PRIMARY KEY(id),
```

```
  INDEX commenter_idx (commenter ASC),
  CONSTRAINT commenter
  FOREIGN KEY (commenter)
  REFERENCES nodejs.users (id)
  ON DELETE CASCADE
  ON UPDATE CASCADE
)
COMMENT='댓글'
DEFAULT CHARSET = utf8mb4
ENGINE = InnoDB;
```

ⓐⓑⓒⓓⓔⓕⓖⓗⓘⓙⓚⓛⓜⓝⓞⓟⓠⓡⓢⓣⓤⓥⓦⓧⓨⓩ

④ models/comment.js

```
const Sequelize = require('sequelize');
module.exports = class Comment extends
Sequelize.Model {
  static init(sequelize) {
    return super.init({
      comment: {
        type: Sequelize.STRING(100),
        allowNull: false,
      },
      created_at: {
        type: Sequelize.DATE,
        allowNull: true,
        defaultValue: Sequelize.NOW,
      },
    }, {
```

```
        sequelize,
        timestamps: false,
        modelName: 'Comment',
        tableName: 'comments',
        paranoid: false,
        charset: 'utf8mb4',
        collate: 'utf8mb4_general_ci',
      });
  }

  static associate(db) {
    db.Comment.belongsTo(db.User, { foreignKey:
    'commenter', targetKey: 'id' });
  }
};
```

ⓐⓑⓒⓓⓔⓕⓖⓗⓘⓙⓚⓛⓜⓝⓞⓟⓠⓡⓢⓣⓤⓥⓦⓧⓨⓩ

(1)	Express 설치
	기본적인 node 서버 설치

Ⓐ npm (node package manager) 초기화

```
npm init -y
```

① package.json

```
"scripts": {
  "start": "node ./dist/index.js",
  "dev": "nodemon ./src/index.ts",
  "build": "rm –rf ./dist/ && tsc -p ."
}
```

Ⓑ express 설치

```
npm install --save express
npm install --save-dev nodemon typescript ts-node @types/node @types/express
```

① ts-node (ts-node-dev)

ⓐ node 상에서, 독립적으로 .ts 파일을 실행한다.

ⓑ node 상에서 TypeScript Compiler를 통하지 않고도, 직접 TypeScript를 실행시키는 역할을 해줌

② @types/??

ⓐ 모듈에 대한 타입을 지원한다.

ⓑ 실제 서버를 구동할때 사용되지 않는다.

③ express - RequestHandler

ⓐ 오류처리를 간단하게 할 수 있는 미들웨어

ⓑ https://lakelouise.tistory.com/227

Ⓒ TypeScript 초기화

```
( npx ) tsc --init
```

① tsconfig.json

```
{
  "compilerOptions": {
    "target": "es6",
    "strict": true,
    "skipLibCheck": true,
    "sourceMap": true,
    "rootDir": "./src",
    "outDir": "./dist",
    "module": "commonjs",
    "moduleResolution": "node",
    "esModuleInterop": true,
  },
  "include": ["src/**/*.ts"],
}
```

Ⓓ 기본 미들웨어 설치

```
npm install --save cors dotenv morgan
npm install --save-dev @types/cors @types/dotenv @types/morgan
```

① cors(Cross-Origin Resource Sharing)

ⓐ 교차 출처 리소스 공유

ⓑ 다른 출처의 리소스를 사용하는 것을 제한하는 행위

② dotenv

ⓐ 환경변수 설정

③ morgan

ⓐ 요청과 응답에 대한 정보를 콘솔에 기록

ⓑ 옵션 : dev(개발), combined(배포), common, shortr, tiny

④ body-parser

ⓐ express.json()

㉠ body parser가 json 데이터를 읽을 수 있도록 정의

ⓑ express.urlencoded({extended: false})

㉠ body parser가 form 데이터를 읽을 수 있도록 정의

㉡ true : Express에 기본 내장된 querystring 모듈(url 주소 뒤에 붙어서 넘어오는 파라미터인 querystring을 쉽게 조작할 수 있는 기능을 제공하는 모듈)을 사용한다.

㉢ false : querystring 모듈의 기능이 좀 더 확장된 qs 모듈을 사용한다. (qs 모듈 별도 설치 필요)

⑤ static

ⓐ app.use(요청경로, 실제경로)

ⓑ 정적 파일들이 담겨 있는 폴더를 지정

⑥ 에러 처리 루틴

```
app.use((err: Error, req: Request, res: Response, next: NextFunction) => {
  res.status(500).json({ message:err.message });
});
```

⑦ 404

```
app.all('*', (req: Request, res:Response) => {
  res.status(404);
  if(req.accepts('html')) {
    res.sendFile(path.join(__dirname, 'viewpage',
    '404.html'))
  } else if(req.accepts('json')) {
        res.json({ error: "404 Not Found" });
  } else {
        res.type('txt').send("404 Not Found")
  }
});
```

ⒺⒻⒼⒽⒾⒿⓀⓁⓂ

(三)	MySQL + Sequelize
	설명

Ⓐ Sequelize 설치

```
npm install --save sequelize mysql2
npm install --save-dev sequelize-cli
```

Ⓑ .sequelizerc 생성

sequelize를 초기화 할 때, 생성되어야 할 파일과 폴더 위치를 지정

```
const path = require('path');

module.exports = {
  'config': path.resolve('src/cfg', 'database.js'),
  'models-path': path.resolve('src/db', 'models'),
  'seeders-path': path.resolve('src/db', 'seeders'),
  'migrations-path': path.resolve('src/db', 'migrations')
}
```

Ⓒ sequelize 초기화

```
npx sequelize-cli init
```

```
Sequelize CLI [Node: 16.15.0, CLI: 6.5.2, ORM: 6.26.0]

Created "src₩config₩dbMySQL.js"
Successfully created models folder at "D:₩WORK₩Express₩express-app₩app
₩src₩dbase₩models".
```

```
Successfully created migrations folder at "D:₩WORK₩Express₩express-app₩app₩src₩dbase₩migrations".
Successfully created seeders folder at "D:₩WORK₩Express₩express-app₩app₩src₩dbase₩seeders".
```

① src/cfg/database.js 파일 수정

```
const path = require('path');
const dotenv = require('dotenv');

dotenv.config({ path: path.join(__dirname, ".env") });

module.exports =
{
  "development": {
    "username": process.env.DB_USER,
    "password": process.env.DB_PASS,
    "database": process.env.DB_NAME,
    "host": process.env.DB_HOST,
    "dialect": "mysql"
  },
  "test": {
    "username": "root",
    "password": null,
    "database": "database_test",
    "host": "127.0.0.1",
    "dialect": "mysql"
  },
  "production": {
    "username": "root",
```

```
    "password": null,
    "database": "database_production",
    "host": "127.0.0.1",
    "dialect": "mysql"
  }
}
```

② src/cfg/dbConnect.ts 생성

```
import path from 'path';
import dotenv from 'dotenv';
import { Sequelize } from 'sequelize';

dotenv.config({ path: path.join(__dirname, ".env") });

const dbHost = process.env.DB_HOST;
const dbName = process.env.DB_NAME as string;
const dbUser = process.env.DB_USER as string;
const dbPass = process.env.DB_PASS;
const dbDialect = 'mysql';

const sequelizeConnection = new Sequelize(dbName,
dbUser, dbPass, {
  host: dbHost,
  dialect: dbDialect
});

export default sequelizeConnection;
```

Ⓓ Model 생성

① Role 모델 생성

models, migrations 폴더에 Role 정의 파일 생성

```
npx sequelize-cli model:generate --name Role --attribu
tes roleName:string,active:boolean
```

New model was created at D:₩WORK₩code₩express-app₩app₩src₩db₩**models**₩role.js.
New migration was created at D:₩WORK₩code₩express-app₩app₩src₩db₩**migrations**₩20221209160642-create-role.js

ⓐ rose.js --> Rose.ts 변경

```
'use strict';
const {
  Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  class Role extends Model {
    /*
      연결을 정의하기 위한 도우미 메서드입니다.
      이 메서드는 Sequelize 수명 주기의 일부가 아닙니다. `models/index` 파일은 이 메소드를 자동으로 호출합니다.
    */
    static associate(models) {
      // 여기에서 연관 정의
    }
  }

  Role.init({
    roleName: DataTypes.STRING,
    active: DataTypes.BOOLEAN
  }, {
```

```
    sequelize,
    modelName: 'Role',
  });
  return Role;
};
```

```
import { DataTypes, Model, Optional } from 'sequelize';
import connection from '../../cfg/dbConnect';

interface RoleAttributes {
  id?: number,
  roleName?: string | null,
  active?: boolean | null,

  createdAt?: Date,
  updateAt?: Date
}

export interface RoleInput extends Optional<RoleAttributes, 'id'>{}
export interface RoleOutput extends Required<RoleAttributes>{}

class Role extends Model<RoleAttributes, RoleInput> implements RoleAttributes {
  public id!: number;
  public roleName!: string | null;
```

```
  public active!: boolean | null;

  public readonly createdAt!: Date;
  public readonly updateAt!: Date;
}

Role.init({
  id: {
    allowNull: false,
    autoIncrement: true,
    primaryKey: true,
    type: DataTypes.BIGINT
  },
  roleName: {
    allowNull: true,
    type: DataTypes.STRING
  },
  active: {
    allowNull: true,
    type: DataTypes.BOOLEAN
  },
  }, {
    timestamps:true,
    sequelize: connection,
    underscored: false
});
```

```
export default Role;
```

ⓑⓒ

② User 모델 생성

models, migrations 폴더에 User 정의 파일 생성

```
npx sequelize-cli model:generate --name User --attributes name:string,email:string,roleId:bigint,password:text,accessToken:text,verified:boolean,active:boolean
```

```
New model was created at D:₩WORK₩code₩express-app₩app₩src₩db₩models₩user.js.
New migration was created at D:₩WORK₩code₩express-app₩app₩src₩db₩migrations₩20221210055537-create-user.js.
```

ⓐ user.js --> User.ts 변경

```
'use strict';
const {
  Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  class User extends Model {
    /*
      연결을 정의하기 위한 도우미 메서드입니다.
      이 메서드는 Sequelize 수명 주기의 일부가 아닙니다. `models/index` 파일은 이 메소드를 자동으로 호출합니다.
```

```
    */
    static associate(models) {
      // 여기에서 연관 정의
    }
  }

  User.init({
    name: DataTypes.STRING,
    email: DataTypes.STRING,
    roleId: DataTypes.BIGINT,
    password: DataTypes.TEXT,
    accessToken: DataTypes.TEXT,
    verified: DataTypes.BOOLEAN,
    active: DataTypes.BOOLEAN
  }, {
    sequelize,
    modelName: 'Role',
  });
  return User;
};
```

```
import { DataTypes, Model, Optional } from 'sequelize';
import connection from '../../cfg/dbConnect';

interface UserAttributes {
   id?: number,
   name?: string | null,
```

```
    email?: string | null,
    roleId?: number,
    password?: string | null,
    accessToken?: string | null,
    verified?: boolean | null,
    active?: boolean | null,

    createdAt?: Date,
    updateAt?: Date
}

export interface UserInput extends Optional<UserAttributes, 'id'>{}
export interface UserOutput extends Required<UserAttributes>{}

class User extends Model<UserAttributes, UserInput> implements UserAttributes {
    public id!: number;
    public name!: string;
    public email!: string;
    public roleId!: number;
    public password!: string;
    public accessToken!: string;
    public verified!: boolean;
    public active!: boolean;
```

```
    public readonly createdAt!: Date;
    public readonly updateAt!: Date;
}

User.init({
    id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: DataTypes.BIGINT
    },
    name: {
        allowNull: true,
        type: DataTypes.STRING
    },
    email: {
        allowNull: true,
        type: DataTypes.STRING
    },
    roleId: {
        allowNull: true,
        type: DataTypes.BIGINT
    },
    password: {
        allowNull: true,
        type: DataTypes.TEXT
    },
```

```
    accessToken: {
      allowNull: true,
      type: DataTypes.TEXT
    },
    verified: {
      allowNull: true,
      type: DataTypes.STRING
    },
    active: {
      allowNull: true,
      type: DataTypes.BOOLEAN
    },
}, {
    timestamps:true,
    sequelize: connection,
    underscored: false
});

export default User;
```

Ⓔ migration (테이블 생성)

```
npx sequelize-cli db:migrate
```

```
Sequelize CLI [Node: 16.15.0, CLI: 6.5.2, ORM: 6.26.0]

Loaded configuration file "src₩cfg₩database.js".
Using environment "development".
```

```
== 20221209160642-create-role: migrating =======
== 20221209160642-create-role: migrated (0.039s)
```

```
Sequelize CLI [Node: 16.15.0, CLI: 6.5.2, ORM: 6.26.0]

Loaded configuration file "src₩cfg₩database.js".
Using environment "development".
== 20221210055536-create-user: migrating =======
== 20221210055536-create-user: migrated (0.023s)
```

Ⓕ Seed 생성 (더미 데이터 넣기)

```
npx sequelize-cli seed:generate --name RoleSeeder
```

```
Sequelize CLI [Node: 16.15.0, CLI: 6.5.2, ORM: 6.26.0]

seeders folder at "D:₩WORK₩code₩express-app₩app₩src₩db₩seeders" already exists.
New seed was created at D:₩WORK₩code₩express-app₩app₩src₩db₩seeders₩20221209164715-RoleSeeder.js .
```

```
npx sequelize-cli seed:generate --name UserSeeder
```

```
Sequelize CLI [Node: 16.15.0, CLI: 6.5.2, ORM: 6.26.0]

seeders folder at "D:₩WORK₩code₩express-app₩app₩src₩db₩seeders" already exists.
New seed was created at D:₩WORK₩code₩express-app₩app₩src₩db₩seeders₩20221209164715-RoleSeeder.js .
```

```
'use strict';
const {
  Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  class Role extends Model {
    /**
     * Helper method for defining associations.
     * This method is not a part of Sequelize lifecycle.
     * The `models/index` file will call this method automatically.
     */
    static associate(models) {
      // define association here
    }
  }
  Role.init({
    roleName: DataTypes.STRING,
    active: DataTypes.BOOLEAN
  }, {
    sequelize,
    modelName: 'Role',
  });
  return Role;
};
```

```
'use strict';

/** @type {import('sequelize-cli').Migration} */
module.exports = {
  async up (queryInterface, Sequelize) {
    /**
     * Add seed commands here.
     *
     * Example:
     * await queryInterface.bulkInsert('People', [{
     *   name: 'John Doe',
     *   isBetaMember: false
     * }], {});
    */
  },

  async down (queryInterface, Sequelize) {
    /**
     * Add commands to revert seed here.
     *
     * Example:
     * await queryInterface.bulkDelete('People', null, {});
     */
  }
};
```

```
'use strict';

/** @type {import('sequelize-cli').Migration} */
module.exports = {
  async up (queryInterface, Sequelize) {
    /**
     * Add seed commands here.
     *
     * Example:
     * await queryInterface.bulkInsert('People', [{
     *    name: 'John Doe',
     *    isBetaMember: false
     * }], {});
    */
  await queryInterface.bulkInsert('Roles',
  [
    {
      roleName: 'Super Admin',
      active: true,
      createdAt: new Date(),
      updatedAt: new Date()
    },
    {
      roleName: 'Admin',
      active: true,
      createdAt: new Date(),
      updatedAt: new Date()
```

```
      },
      {
        roleName: 'User',
        active: true,
        createdAt: new Date(),
        updatedAt: new Date()
      }
    ], {})
  },

  async down (queryInterface, Sequelize) {
    /**
     * Add commands to revert seed here.
     *
     * Example:
     * await queryInterface.bulkDelete('People', null, {});
     */
    await queryInterface.bulkDelete('Roles', null, {})
  }
};
```

① 20221209164715-RoleSeeder.js 변경

⑥ Seed 실행

```
npx sequelize-cli db:seed:all
```

```
Sequelize CLI [Node: 16.15.0, CLI: 6.5.2, ORM: 6.26.0]

Loaded configuration file "src₩cfg₩database.js".
Using environment "development".
== 20221209164715-RoleSeeder: migrating =======
== 20221209164715-RoleSeeder: migrated (0.012s)
```

==========

⑴⑵⑶⑷⑸⑹⑺⑻⑼⑽⑾

⑿⒀⒁⒂⒃⒄⒅⒆⒇⒜

㈡㈢㈣㈥㈦㈨㈩◎

ⒶⒷⒸⒹⒺⒻⒼⒽⒾⒿⓀⓁⓂⓃⓄⓅⓆⓇⓈⓉⓊⓋⓌⓍⓎⓏ

①②③④⑤⑥⑦⑧⑨⑩⑪⑫⑬⑭⑮⑯⑰⑱⑲⑳㉑㉒㉓㉔㉕㉖㉗㉘㉙㉚

ⓐⓑⓒⓓⓔⓕⓖⓗⓘⓙⓚⓛⓜⓝⓞⓟⓠⓡⓢⓣⓤⓥⓦⓧⓨⓩ

㊀㊁㊂㊃㊄㊅㊆㊇㊈㊉

• ○ ✪ ↳ · ▸ ↱ ↳ ⇨ ❝ ❞ ❛ ❜ ⇓ · → ⇦

==========

㈢	APPENDIX
	설명