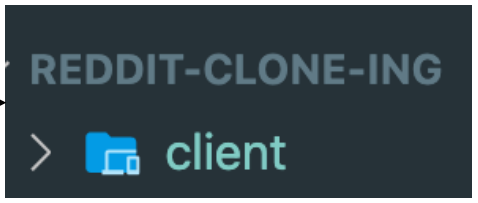
Next JS Typescript 를 이용한 프론트엔드 구현하기

폴더 생성



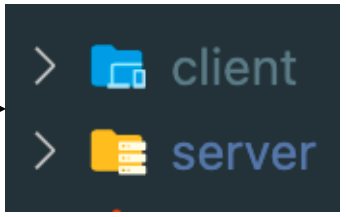
nextjs 앱 설치

```
npx create-next-app@latest --typescript client
```

nextjs 앱 구조

Node.js Express Typescript 를 이용한 백엔드 구현하기

폴더 생성



npm init

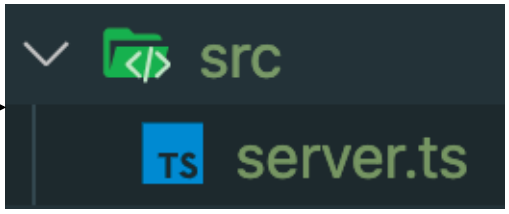
package.json 파일 생성합니다.

프로젝트 정보와 의존성(dependencies)을 관리하는 문서입니다.

패키지 설치 시 여기에 그 정보가 기록됩니다 (이름, 버전등)

package.json
main 파일 변경

main 파일 생성
(시작점)



필요한 모듈 설치

```
npm install morgan nodemon express —save
```

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

* **nodemon**

서버 코드를 변경 할 때마다 서버를 재시작하 일을 자동으로 대신 해줍니다.

* **ts-node**

Node.js 상에서 TypeScript Compiler를 통하지 않고도, 직접 TypeScript를 실행시키는 역할을 합니다.

* **morgan**

nodeJS 에서 사용되는 로그 관리를 위한 미들웨어 입니다.

* **@types/express @types/node**

Express 및 NodeJS에 대한 Type 정의에 도움이 됩니다..

tsconfig.json 파일 생성

TypeScript로 짜여진 코드를 JavaScript로 컴파일하는 옵션을 설정하는 파일입니다. TypeScript 컴파일은 tsc 라는 명령어를 사용합니다.
아래 커맨드로 tsconfig.json 파일을 생성합니다.

```
npx tsc --init
```

옵션에 대한 설명들
https://www.typescriptlang.org/tsconfig

```
{
"compilerOptions": {
/* https://aka.ms/tsconfig.json 를 방문하면 해당 파일에 대한 더 많은 정보를 얻을 수 있습니다. */
// 옵션은 아래와 같은 형식으로 구성되어 있습니다.
// "모듈 키": 모듈 값 /* 설명: 사용가능 옵션 (설명이 "~ 여부"인 경우 'true', 'false') */
/* 기본 옵션 */
// "incremental": true, /* 증분 컴파일 설정 여부 */
"target": "es5", /* 사용할 특정 ECMAScript 버전 설정: 'ES3' (기본), 'ES5', 'ES2015', 'ES2016', 'ES2017', 'ES2018',
'ES2019', 'ES2020', 혹은 'ESNEXT'. */
"module": "commonjs", /* 모듈을 위한 코드 생성 설정: 'none', 'commonjs', 'amd', 'system', 'umd', 'es2015', 'es2020',
or 'ESNext'. */
// "lib": [], /* 컴파일에 포함될 라이브러리 파일 목록 */
// "allowJs": true, /* 자바스크립트 파일 컴파일 허용 여부 */
// "checkJs": true, /* .js 파일의 오류 검사 여부 */
// "jsx": "preserve", /* JSX 코드 생성 설정: 'preserve', 'react-native', 혹은 'react'. */
// "declaration": true, /* '.d.ts' 파일 생성 여부. */
// "declarationMap": true, /* 각 '.d.ts' 파일의 소스맵 생성 여부. */
// "sourceMap": true, /* '.map' 파일 생성 여부. */
```

```
// "outFile": "./", /* 단일 파일로 합쳐서 출력합니다. */
// "outDir": "./", /* 해당 디렉토리로 결과 구조를 보냅니다. */
// "rootDir": "./", /* 입력 파일의 루트 디렉토리(rootDir) 설정으로 --outDir로 결과 디렉토리 구조를 조작할 때 사용됩니다. */
// "composite": true, /* 프로젝트 컴파일 여부 */
// "tsBuildInfoFile": "./", /* 증분 컴파일 정보를 저장할 파일 */
// "removeComments": true, /* 주석 삭제 여부 */
// "noEmit": true, /* 결과 파일 내보낼지 여부 */
// "importHelpers": true, /* 'tslib'에서 헬퍼를 가져올 지 여부 */
// "downlevelIteration": true, /* 타겟이 'ES5', 'ES3'일 때에도 'for-of', spread 그리고 destructuring 문법 모두 지원
*/
// "isolatedModules": true, /* 각 파일을 분리된 모듈로 트랜스파일 ('ts.transpileModule'과 비슷합니다). */
/* 엄격한 타입-확인 옵션 */
"strict": true, /* 모든 엄격한 타입-체킹 옵션 활성화 여부 */
// "noImplicitAny": true, /* 'any' 타입으로 구현된 표현식 혹은 정의 에러처리 여부 */
// "strictNullChecks": true, /* 엄격한 null 확인 여부 */
// "strictFunctionTypes": true, /* 함수 타입에 대한 엄격한 확인 여부 */
// "strictBindCallApply": true, /* 함수에 엄격한 'bind', 'call' 그리고 'apply' 메소드 사용 여부 */
// "strictPropertyInitialization": true, /* 클래스의 값 초기화에 엄격한 확인 여부 */
// "noImplicitThis": true, /* 'any' 타입으로 구현된 'this' 표현식 에러처리 여부 */
// "alwaysStrict": true, /* strict mode로 분석하고 모든 소스 파일에 "use strict"를 추가할 지 여부 */
/* 추가적인 확인 */
// "noUnusedLocals": true, /* 사용되지 않은 지역 변수에 대한 에러보고 여부 */
// "noUnusedParameters": true, /* 사용되지 않은 파라미터에 대한 에러보고 여부 */
// "noImplicitReturns": true, /* 함수에서 코드의 모든 경로가 값을 반환하지 않을 시 에러보고 여부 */
// "noFallthroughCasesInSwitch": true, /* switch문에서 fallthrough 케이스에 대한 에러보고 여부 */
/* 모듈 해석 옵션 */
// "moduleResolution": "node", /* 모듈 해석 방법 설정: 'node' (Node.js) 혹은 'classic' (TypeScript pre-1.6). */
// "baseUrl": "./", /* non-absolute한 모듈 이름을 처리할 기준 디렉토리 */
// "paths": {}, /* 'baseUrl'를 기준으로 불러올 모듈의 위치를 재지정하는 엔트리 시리즈 */
// "rootDirs": [], /* 결합된 컨텐츠가 런타임에서의 프로젝트 구조를 나타내는 루트 폴더들의 목록 */
// "typeRoots": [], /* 타입 정의를 포함할 폴더 목록, 설정 안 할 시 기본적으로 ./node_modules/@types로 설정 */
// "types": [], /* 컴파일중 포함될 타입 정의 파일 목록 */
// "allowSyntheticDefaultImports": true, /* default export이 아닌 모듈에서도 default import가 가능하게 할 지 여부, 해
당 설정은 코드 추출에 영향은 주지 않고, 타입확인에만 영향을 줍니다. */
"esModuleInterop": true, /* 모든 imports에 대한 namespace 생성을 통해 CommonJS와 ES Modules 간의 상호 운용성이 생기게할
지 여부, 'allowSyntheticDefaultImports'를 암시적으로 승인합니다. */
// "preserveSymlinks": true, /* symlik의 실제 경로를 처리하지 않을 지 여부 */
// "allowUmdGlobalAccess": true, /* UMD 전역을 모듈에서 접근할 수 있는 지 여부 */
/* 소스 맵 옵션 */
// "sourceRoot": "", /* 소스 위치 대신 디버거가 알아야 할 TypeScript 파일이 위치할 곳 */
// "mapRoot": "", /* 생성된 위치 대신 디버거가 알아야 할 맵 파일이 위치할 곳 */
// "inlineSourceMap": true, /* 분리된 파일을 가지고 있는 대신, 단일 파일을 소스 맵과 가지고 있을 지 여부 */
// "inlineSources": true, /* 소스맵과 나란히 소스를 단일 파일로 내보낼 지 여부, '--inlineSourceMap' 혹은 '--sourceMap'가
설정되어 있어야 한다. */
/* 실험적 옵션 */
// "experimentalDecorators": true, /* ES7의 decorators에 대한 실험적 지원 여부 */
// "emitDecoratorMetadata": true, /* decorator를 위한 타입 메타데이터를 내보내는 것에 대한 실험적 지원 여부 */
/* 추가적 옵션 */
"skipLibCheck": true, /* 정의 파일의 타입 확인을 건너 뛸 지 여부 */
"forceConsistentCasingInFileNames": true /* 같은 파일에 대한 일관되지 않은 참조를 허용하지 않을 지 여부 */
}
}
```

express 코드 작성

root 폴더 아래 src/server.ts를 작성해줍니다.

```
import express from "express";
import morgan from "morgan";
```

```
const app = express();

app.use(express.json());
app.use(morgan("dev"));
app.get("/", (_, res) => res.send("running"));

let port = 4000;
app.listen(port, async () => {
    console.log(`Server running at http://localhost:${port}`);
});
```

- dev
- short
- common
- combined

app.get의 url로 접속을 하면 해당 블록의 코드를 실행합니다.

app.listen의 포트로 접속하면 해당 블록의 코드를 실행합니다.

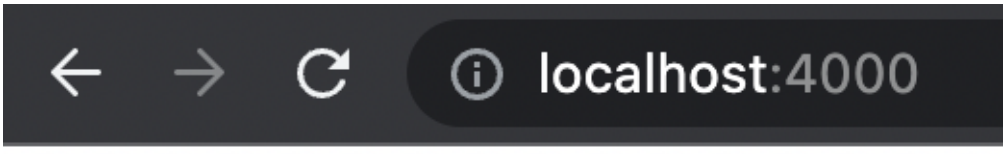
package.json 파일 수정

package.json 파일을 수정합니다.

아까 설치한 nodemon과 ts-node를 이용하여 서버를 시작하는 스크립트를 작성합니다.

```
"scripts": {
  "start": "ts-node src/server.ts",
  "dev": "nodemon --exec ts-node ./src/server.ts",
  "test": "echo \"Error: no test specified\" && exit 1"
},
```

백엔드 서버 실행 후 접속

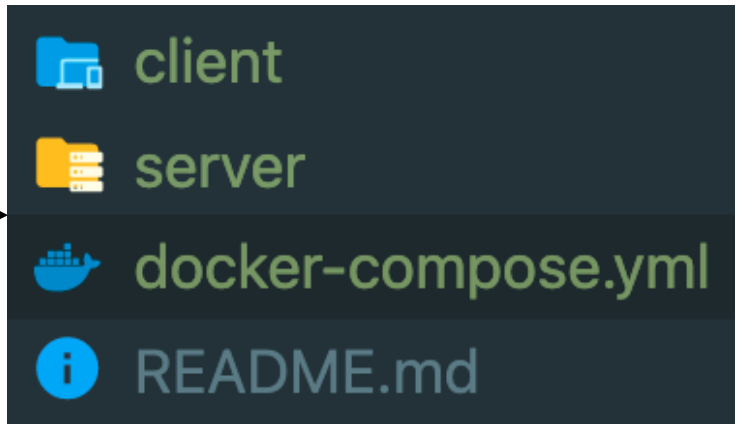


running

도커를 이용한 Postgres 실행

도커 설치 및 실행

docker-compse.yml 파일 생성



docker-compse.yml 파일 작성

```yaml
version: "3"
services:
    db:
        image: postgres:latest
        container_name: postgres
        restart: always
        ports:
            - "5432:5432"
        environment:
            POSTGRES_USER: "postgres"
            POSTGRES_PASSWORD: "password"
        volumes:
            - ./data:/var/lib/postgresql/data
```

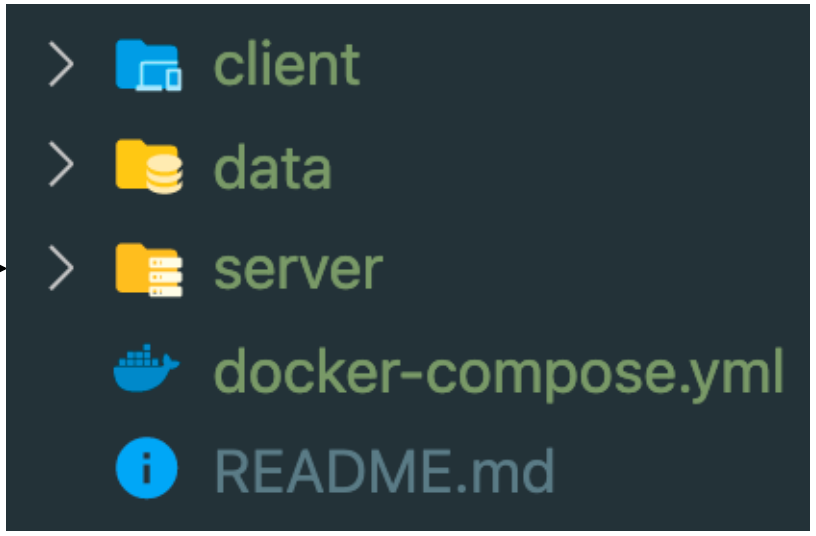
환경 변수 이용

```yaml
version: "3"
services:
    db:
        image: postgres:latest
        container_name: postgres
        restart: always
        ports:
            - "5432:5432"
        environment:
            POSTGRES_USER: "${DB_USER_ID}"
            POSTGRES_PASSWORD: "${DB_USER_PASSWORD}"
        volumes:
            - ./data:/var/lib/postgresql/data
```

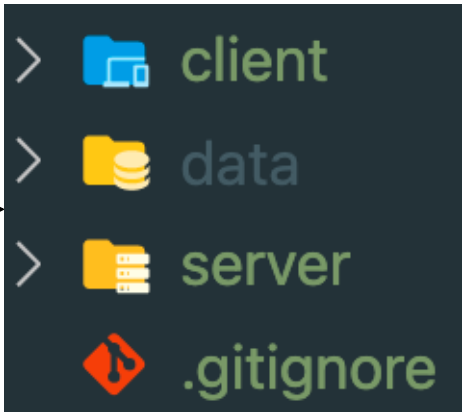
.env

```
DB_USER_ID=postgres
DB_USER_PASSWORD=password
```

docker-compose up
명령어로
도커 컨테이너 실행



.gitignore 파일 생성



.gitignore

```
1 data
```

데이터베이스와 애플리케이션 연결

https://typeorm.io/

필요한 모듈 설치

https://typeorm.io/#installation

`npm install pg typeorm reflect-metadata —save`

* pg

PostgreSQL 데이터베이스와 인터페이스하기 위한 NodeJS 모듈 모음입니다.

인터페이스
일련의 명령어나 함수, 옵션, 그리고 프로그램 언어에 의해 제공되는 명령어나 데이터를 표현하기 위한 다른 방법들로 구성되는 프로그래밍 인터페이스
==> 간단하게 postgres 데이터베이스를 노드js에서 사용할 수 있게 도와주는 모듈. callbacks, promises, connection pooling, C/C++ bindings 등등

* typeorm

TypeScript 및 JavaScript(ES5, ES6, ES7, ES8)와 함께 사용할 수 있는 Node JS에서 실행되는 ORM입니다.

* reflect-metadata

With the reflect-metadata package you can do runtime reflection on types. Since TypeORM mostly works with decorators (like @Entity or @Column), this package is used to parse these decorators and use it for building sql queries. https://stackoverflow.com/a/49639117

The name reflection is used to describe code which is able to inspect other code in the same system (or itself).

typeorm 설정 파일 생성

npx typeorm init

```
MyProject
├── src                  // place of your TypeScript code
│   ├── entity           // place where your entities (database models) are stored
│   │   └── User.ts      // sample entity
│   ├── migration        // place where your migrations are stored
│   ├── data-source.ts   // data source and all connection configuration
│   └── index.ts         // start point of your application
├── .gitignore           // standard gitignore file
├── package.json         // node module dependencies
├── README.md            // simple readme file
```

```
└── tsconfig.json          // TypeScript compiler option
```

typeorm 설정 파일 작성

```typescript
import "reflect-metadata"
import { DataSource } from "typeorm"

export const AppDataSource = new DataSource({
    type: "postgres",
    host: "localhost",
    port: 5432,
    username: "postgres",
    password: "password",
    database: "postgres",
    synchronize: true,
    logging: false,
    entities: [
        "src/entities/**/*.ts"
    ],
    migrations: [],
    subscribers: [],
})
```

백엔드 실행 시 데이터베이스 연결

```typescript
// to initialize initial connection with the database, register all entities
// and "synchronize" database schema, call "initialize()" method of a newly created database
// once in your application bootstrap
AppDataSource.initialize()
.then(() => {
// here you can start to work with your database
console.log('initialized')
})
.catch((error) => console.log(error))
```

백엔드 실행
npm run dev

```
Server running at http://localhost:4000
query: SELECT * FROM current_schema()
query: SHOW server_version;
query: START TRANSACTION
query: SELECT * FROM "information_schema
query: CREATE TABLE "typeorm_metadata" (
ng, "table" character varying, "name" ch
query: SELECT * FROM "information_schema
query: SELECT * FROM current_database()
query: SELECT * FROM current_schema()
query: SELECT "t".* FROM "typeorm_metada
pg_namespace" "n" ON "n"."oid" = "c"."re
IEW') AND (1=1)
query: COMMIT
Datebase connected!
```

만약 다음과 같은 에러가 난다면

postgres | 2022-02-26 01:26:22.922 UTC [35] FATAL: no pg_hba.conf entry for host "172.23.0.1", user "postgres", database "postgres", no encryption

```
postgres | 2022-02-26 01:26:22.922 UTC [35] FATAL:  no pg_hba.conf entr
y for host "172.23.0.1", user "postgres", database "postgres", no encry
ption
```

data 폴더 임의로 지운 후 다시 docker-compose up -build 로 다시 실행

```
{
"compilerOptions": {
/* Visit https://aka.ms/tsconfig.json to read more about this file */

/* Projects */
// "incremental": true, /* Enable incremental compilation */
// "composite": true, /* Enable constraints that allow a TypeScript project
to be used with project references. */
// "tsBuildInfoFile": "./", /* Specify the folder for .tsbuildinfo
incremental compilation files. */
// "disableSourceOfProjectReferenceRedirect": true, /* Disable preferring
source files instead of declaration files when referencing composite
projects */
// "disableSolutionSearching": true, /* Opt a project out of multi-project
reference checking when editing. */
// "disableReferencedProjectLoad": true, /* Reduce the number of projects
loaded automatically by TypeScript. */

/* Language and Environment */
"target": "es2016", /* Set the JavaScript language version for emitted
JavaScript and include compatible library declarations. */
// "lib": [], /* Specify a set of bundled library declaration files that
describe the target runtime environment. */
// "jsx": "preserve", /* Specify what JSX code is generated. */
"experimentalDecorators": true, /* Enable experimental support for TC39
stage 2 draft decorators. */
"emitDecoratorMetadata": true, /* Emit design-type metadata for decorated
declarations in source files. */
// "jsxFactory": "", /* Specify the JSX factory function used when targeting
React JSX emit, e.g. 'React.createElement' or 'h' */
// "jsxFragmentFactory": "", /* Specify the JSX Fragment reference used for
fragments when targeting React JSX emit e.g. 'React.Fragment' or 'Fragment'.
*/
// "jsxImportSource": "", /* Specify module specifier used to import the JSX
factory functions when using `jsx: react-jsx*`.` */
// "reactNamespace": "", /* Specify the object invoked for `createElement`.
This only applies when targeting `react` JSX emit. */
// "noLib": true, /* Disable including any library files, including the
default lib.d.ts. */
// "useDefineForClassFields": true, /* Emit ECMAScript-standard-compliant
class fields. */
```

```
/* Modules */
"module": "commonjs", /* Specify what module code is generated. */
// "rootDir": "./", /* Specify the root folder within your source files. */
// "moduleResolution": "node", /* Specify how TypeScript looks up a file
from a given module specifier. */
// "baseUrl": "./", /* Specify the base directory to resolve non-relative
module names. */
// "paths": {}, /* Specify a set of entries that re-map imports to
additional lookup locations. */
// "rootDirs": [], /* Allow multiple folders to be treated as one when
resolving modules. */
// "typeRoots": [], /* Specify multiple folders that act like
`./node_modules/@types`. */
// "types": [], /* Specify type package names to be included without being
referenced in a source file. */
// "allowUmdGlobalAccess": true, /* Allow accessing UMD globals from
modules. */
// "resolveJsonModule": true, /* Enable importing .json files */
// "noResolve": true, /* Disallow `import`s, `require`s or `<reference>`s
from expanding the number of files TypeScript should add to a project. */

/* JavaScript Support */
// "allowJs": true, /* Allow JavaScript files to be a part of your program.
Use the `checkJS` option to get errors from these files. */
// "checkJs": true, /* Enable error reporting in type-checked JavaScript
files. */
// "maxNodeModuleJsDepth": 1, /* Specify the maximum folder depth used for
checking JavaScript files from `node_modules`. Only applicable with
`allowJs`. */

/* Emit */
// "declaration": true, /* Generate .d.ts files from TypeScript and
JavaScript files in your project. */
// "declarationMap": true, /* Create sourcemaps for d.ts files. */
// "emitDeclarationOnly": true, /* Only output d.ts files and not JavaScript
files. */
// "sourceMap": true, /* Create source map files for emitted JavaScript
files. */
// "outFile": "./", /* Specify a file that bundles all outputs into one
JavaScript file. If `declaration` is true, also designates a file that
bundles all .d.ts output. */
// "outDir": "./", /* Specify an output folder for all emitted files. */
// "removeComments": true, /* Disable emitting comments. */
// "noEmit": true, /* Disable emitting files from a compilation. */
// "importHelpers": true, /* Allow importing helper functions from tslib
once per project, instead of including them per-file. */
// "importsNotUsedAsValues": "remove", /* Specify emit/checking behavior for
imports that are only used for types */
// "downlevelIteration": true, /* Emit more compliant, but verbose and less
performant JavaScript for iteration. */
// "sourceRoot": "", /* Specify the root path for debuggers to find the
reference source code. */
// "mapRoot": "", /* Specify the location where debugger should locate map
files instead of generated locations. */
// "inlineSourceMap": true, /* Include sourcemap files inside the emitted
JavaScript. */
// "inlineSources": true, /* Include source code in the sourcemaps inside
the emitted JavaScript. */
// "emitBOM": true, /* Emit a UTF-8 Byte Order Mark (BOM) in the beginning
of output files. */
// "newLine": "crlf", /* Set the newline character for emitting files. */
// "stripInternal": true, /* Disable emitting declarations that have
`@internal` in their JSDoc comments. */
// "noEmitHelpers": true, /* Disable generating custom helper functions like
`__extends` in compiled output. */
```

```
// "noEmitOnError": true, /* Disable emitting files if any type checking
errors are reported. */
// "preserveConstEnums": true, /* Disable erasing `const enum` declarations
in generated code. */
// "declarationDir": "./", /* Specify the output directory for generated
declaration files. */
// "preserveValueImports": true, /* Preserve unused imported values in the
JavaScript output that would otherwise be removed. */

/* Interop Constraints */
// "isolatedModules": true, /* Ensure that each file can be safely
transpiled without relying on other imports. */
// "allowSyntheticDefaultImports": true, /* Allow 'import x from y' when a
module doesn't have a default export. */
"esModuleInterop": true, /* Emit additional JavaScript to ease support for
importing CommonJS modules. This enables `allowSyntheticDefaultImports` for
type compatibility. */
// "preserveSymlinks": true, /* Disable resolving symlinks to their
realpath. This correlates to the same flag in node. */
"forceConsistentCasingInFileNames": true, /* Ensure that casing is correct
in imports. */

/* Type Checking */
"strict": false, /* Enable all strict type-checking options. */
// "noImplicitAny": true, /* Enable error reporting for expressions and
declarations with an implied `any` type.. */
// "strictNullChecks": true, /* When type checking, take into account `null`
and `undefined`. */
// "strictFunctionTypes": true, /* When assigning functions, check to ensure
parameters and the return values are subtype-compatible. */
// "strictBindCallApply": true, /* Check that the arguments for `bind`,
`call`, and `apply` methods match the original function. */
"strictPropertyInitialization": false, /* Check for class properties that
are declared but not set in the constructor. */
// "noImplicitThis": true, /* Enable error reporting when `this` is given
the type `any`. */
// "useUnknownInCatchVariables": true, /* Type catch clause variables as
'unknown' instead of 'any'. */
// "alwaysStrict": true, /* Ensure 'use strict' is always emitted. */
// "noUnusedLocals": true, /* Enable error reporting when a local variables
aren't read. */
// "noUnusedParameters": true, /* Raise an error when a function parameter
isn't read */
// "exactOptionalPropertyTypes": true, /* Interpret optional property types
as written, rather than adding 'undefined'. */
// "noImplicitReturns": true, /* Enable error reporting for codepaths that
do not explicitly return in a function. */
// "noFallthroughCasesInSwitch": true, /* Enable error reporting for
fallthrough cases in switch statements. */
// "noUncheckedIndexedAccess": true, /* Include 'undefined' in index
signature results */
// "noImplicitOverride": true, /* Ensure overriding members in derived
classes are marked with an override modifier. */
// "noPropertyAccessFromIndexSignature": true, /* Enforces using indexed
accessors for keys declared using an indexed type */
// "allowUnusedLabels": true, /* Disable error reporting for unused labels.
*/
// "allowUnreachableCode": true, /* Disable error reporting for unreachable
code. */

/* Completeness */
// "skipDefaultLibCheck": true, /* Skip type checking .d.ts files that are
included with TypeScript. */
"skipLibCheck": true /* Skip type checking all .d.ts files. */
}
```

```
}


// {
// "compilerOptions": {
// /* https://aka.ms/tsconfig.json 를 방문하면 해당 파일에 대한 더 많은 정보를 얻을 수 있습니다. */
// // 옵션은 아래와 같은 형식으로 구성되어 있습니다.
// // "모듈 키": 모듈 값 /* 설명: 사용가능 옵션 (설명이 "~ 여부"인 경우 'true', 'false') */
// /* 기본 옵션 */
// // "incremental": true, /* 증분 컴파일 설정 여부 */
// "target": "es5", /* 사용할 특정 ECMAScript 버전 설정: 'ES3' (기본), 'ES5', 'ES2015', 'ES2016', 'ES2017', 'ES2018', 'ES2019', 'ES2020', 혹은 'ESNEXT'. */
// "module": "commonjs", /* 모듈을 위한 코드 생성 설정: 'none', 'commonjs', 'amd', 'system', 'umd', 'es2015', 'es2020', or 'ESNext'. */
// // "lib": [], /* 컴파일에 포함될 라이브러리 파일 목록 */
// // "allowJs": true, /* 자바스크립트 파일 컴파일 허용 여부 */
// // "checkJs": true, /* .js 파일의 오류 검사 여부 */
// // "jsx": "preserve", /* JSX 코드 생성 설정: 'preserve', 'react-native', 혹은 'react'. */
// // "declaration": true, /* '.d.ts' 파일 생성 여부. */
// // "declarationMap": true, /* 각 '.d.ts' 파일의 소스맵 생성 여부. */
// // "sourceMap": true, /* '.map' 파일 생성 여부. */
// // "outFile": "./", /* 단일 파일로 합쳐서 출력합니다. */
// // "outDir": "./", /* 해당 디렉토리로 결과 구조를 보냅니다. */
// // "rootDir": "./", /* 입력 파일의 루트 디렉토리(rootDir) 설정으로 --outDir로 결과 디렉토리 구조를 조작할 때 사용됩니다. */
// // "composite": true, /* 프로젝트 컴파일 여부 */
// // "tsBuildInfoFile": "./", /* 증분 컴파일 정보를 저장할 파일 */
// // "removeComments": true, /* 주석 삭제 여부 */
// // "noEmit": true, /* 결과 파일 내보낼지 여부 */
// // "importHelpers": true, /* 'tslib'에서 헬퍼를 가져올 지 여부 */
// // "downlevelIteration": true, /* 타겟이 'ES5', 'ES3'일 때에도 'for-of', spread 그리고 destructuring 문법 모두 지원 */
// // "isolatedModules": true, /* 각 파일을 분리된 모듈로 트랜스파일 ('ts.transpileModule'과 비슷합니다). */
// /* 엄격한 타입-확인 옵션 */
// "strict": true, /* 모든 엄격한 타입-체킹 옵션 활성화 여부 */
// // "noImplicitAny": true, /* 'any' 타입으로 구현된 표현식 혹은 정의 에러처리 여부 */
// // "strictNullChecks": true, /* 엄격한 null 확인 여부 */
// // "strictFunctionTypes": true, /* 함수 타입에 대한 엄격한 확인 여부 */
// // "strictBindCallApply": true, /* 함수에 엄격한 'bind', 'call' 그리고 'apply' 메소드 사용 여부 */
// // "strictPropertyInitialization": true, /* 클래스의 값 초기화에 엄격한 확인 여부 */
// // "noImplicitThis": true, /* 'any' 타입으로 구현된 'this' 표현식 에러처리 여부 */
// // "alwaysStrict": true, /* strict mode로 분석하고 모든 소스 파일에 "use strict"를 추가할 지 여부 */
// /* 추가적인 확인 */
// // "noUnusedLocals": true, /* 사용되지 않은 지역 변수에 대한 에러보고 여부 */
// // "noUnusedParameters": true, /* 사용되지 않은 파라미터에 대한 에러보고 여부 */
// // "noImplicitReturns": true, /* 함수에서 코드의 모든 경로가 값을 반환하지 않을 시 에러보고 여부 */
// // "noFallthroughCasesInSwitch": true, /* switch문에서 fallthrough 케이스에 대한 에러보고 여부 */
// /* 모듈 해석 옵션 */
// // "moduleResolution": "node", /* 모듈 해석 방법 설정: 'node' (Node.js) 혹은 'classic' (TypeScript pre-1.6). */
// // "baseUrl": "./", /* non-absolute한 모듈 이름을 처리할 기준 디렉토리 */
// // "paths": {}, /* 'baseUrl'를 기준으로 불러올 모듈의 위치를 재지정하는 엔트리
```

```
시리즈 */
// // "rootDirs": [], /* 결합된 컨텐츠가 런타임에서의 프로젝트 구조를 나타내는 루트 폴더들의 목록 */
// // "typeRoots": [], /* 타입 정의를 포함할 폴더 목록, 설정 안 할 시 기본적으로 ./node_modules/@types로 설정 */
// // "types": [], /* 컴파일중 포함될 타입 정의 파일 목록 */
// // "allowSyntheticDefaultImports": true, /* default export이 아닌 모듈에서도 default import가 가능하게 할 지 여부, 해당 설정은 코드 추출에 영향은 주지 않고, 타입확인에만 영향을 줍니다. */
// "esModuleInterop": true, /* 모든 imports에 대한 namespace 생성을 통해 CommonJS와 ES Modules 간의 상호 운용성이 생기게할 지 여부, 'allowSyntheticDefaultImports'를 암시적으로 승인합니다. */
// // "preserveSymlinks": true, /* symlik의 실제 경로를 처리하지 않을 지 여부 */
// // "allowUmdGlobalAccess": true, /* UMD 전역을 모듈에서 접근할 수 있는 지 여부 */
// /* 소스 맵 옵션 */
// // "sourceRoot": "", /* 소스 위치 대신 디버거가 알아야 할 TypeScript 파일이 위치할 곳 */
// // "mapRoot": "", /* 생성된 위치 대신 디버거가 알아야 할 맵 파일이 위치할 곳 */
// // "inlineSourceMap": true, /* 분리된 파일을 가지고 있는 대신, 단일 파일을 소스맵과 가지고 있을 지 여부 */
// // "inlineSources": true, /* 소스맵과 나란히 소스를 단일 파일로 내보낼 지 여부, '--inlineSourceMap' 혹은 '--sourceMap'가 설정되어 있어야 한다. */
// /* 실험적 옵션 */
// // "experimentalDecorators": true, /* ES7의 decorators에 대한 실험적 지원 여부 */
// // "emitDecoratorMetadata": true, /* decorator를 위한 타입 메타데이터를 내보내는 것에 대한 실험적 지원 여부 */
// /* 추가적 옵션 */
// "skipLibCheck": true, /* 정의 파일의 타입 확인을 건너 뛸 지 여부 */
// "forceConsistentCasingInFileNames": true /* 같은 파일에 대한 일관되지 않은 참조를 허용하지 않을 지 여부 */
// }
// }
```

TypeORM (Object Relational Mapping)소개

TypeORM이란?

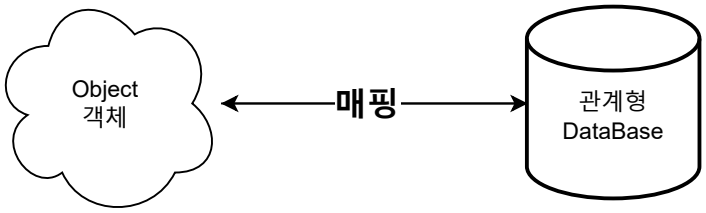
TypeORM은 node.js에서 실행되고 TypeScript로 작성된 객체 관계형 매퍼 라이브러리입니다.

TypeORM은 MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, Oracle, SAP Hana 및 WebSQL과 같은 여러 데이터베이스를 지원합니다.

ORM (Object Relational Mapping) 이란?

객체와 관계형 데이터베이스의 데이터를 자동으로 변형 및 연결하는 작업입니다.

ORM을 이용한 개발은 객체와 데이터베이스의 변형에 유연하게 사용할 수 있습니다.



객체지향 프로그래밍은 클래스 사용

관계형 데이터베이스는 테이블 사용

객체 모델과 관계형 모델 간 불일치 존재...

TypeORM vs Pure Javascript

```
const boards = Board.find({ title: 'Hello' , status: 'PUBLIC' });
```

```
db.query('SELECT * FROM boards WHERE title = "Hello" AND status = "PUBLIC" , (err, result) =>
{
```

```
    if(err) {
      throw new Error('Error')
    }
    boards = result.rows;
})
```

TypeORM 특징과 이점

- 모델을 기반으로 데이터베이스 테이블 체계를 자동으로 생성합니다.
- 데이터베이스에서 개체를 쉽게 삽입, 업데이트 및 삭제할 수 있습니다.
- 테이블 간의 매핑 (일대일, 일대 다 및 다 대다)을 만듭니다.
- 간단한 CLI 명령을 제공합니다.

- TypeORM은 간단한 코딩으로 ORM 프레임 워크를 사용하기 쉽습니다.
- TypeORM은 다른 모듈과 쉽게 통합됩니다.