PostgresSQL 설치하기

지금까지는 NestJS 기능을 익히는데 더 집중을 하기 위해서 데이터를 보관하는데 있어서 메모리를 사용하였습니다. 이제 부터는 애플리케이션에 데이터베이스를 연결해서 데이터베이스에 보관하겠습니다. 데이터베이스는 Postgres를 사용하겠습니다.

설치할 두가지

1. PostgresSQL

2. pgAdmin (데이터베이스를 보는 툴(Tool)입니다.)

Window 에서 PostgresSQL 설치하기

이 사이트 가서 인스톨러 다운로드

https://www.postgresql.org/download/windows/

Mac 에서 PostgresSQL 설치하기

이 사이트 가서 인스톨러 다운로드

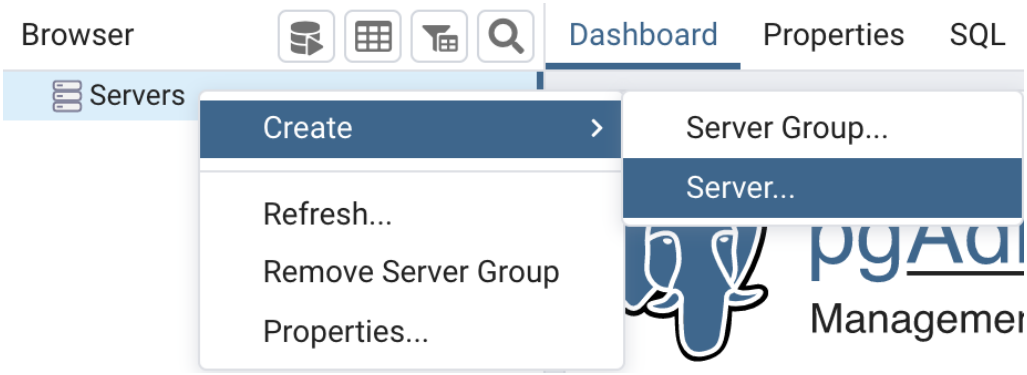
https://postgresapp.com/downloads.html

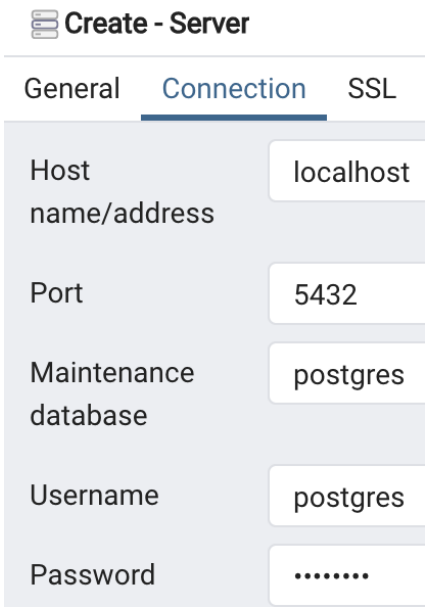
Window & Mac 에서 pgAdmin 설치하기

이 사이트 가서 인스톨러 다운로드

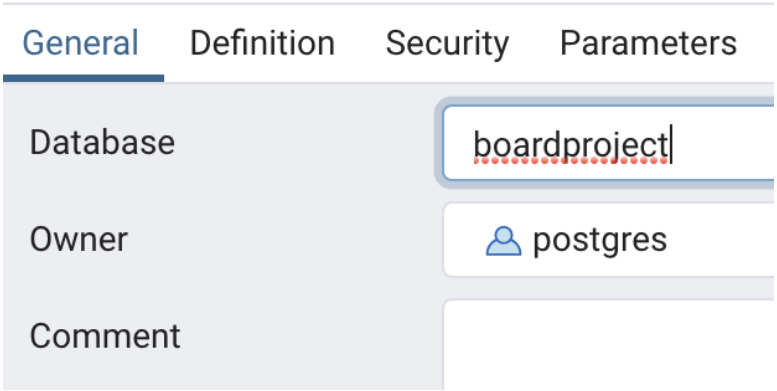
https://www.pgadmin.org/download/

Database 생성





기본 비밀 번호 postgres



TypeORM (Object Relational Mapping)소개

TypeORM이란?

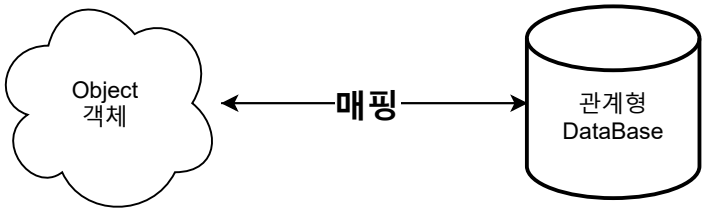
TypeORM은 node.js에서 실행되고 TypeScript로 작성된 객체 관계형 매퍼 라이브러리입니다.

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) {
```

```
    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.

TypeORM 애플리케이션에서 이용하기

TypeORM을 사용하기 위해서 설치해야하는 모듈들

@nestjs/typeorm
- NestJS에서 TypeOrm을 사용하기 위해 연동시켜주는 모듈

typeorm
- TypeORM 모듈

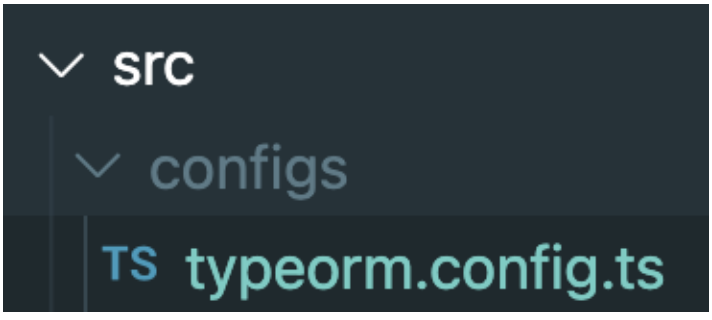
pg
- Postgres 모듈

npm install pg typeorm @nestjs/typeorm --save

다큐멘테이션 :
https://docs.nestjs.com/techniques/database

TypeORM 애플리케이션에 연결하기

1.typeORM 설정파일 생성



2.typeORM 설정파일 작성

- synchronize true는 production 모드에서는 false로... 그렇지 않을시 데이터를 잃을수 있습니다.

```
export const typeORMConfig : TypeOrmModuleOptions = {
    //Database Tyoe
    type:'postgres',
```

```
    host: 'localhost',
    port: 5432,
    username: 'postgres',
    password: 'postgres',
    database: 'boardproject',
    //Entities to be loaded for this connection
    entities: [__dirname + '/../**/*.entity.{js,ts}'],
    // Indicates if database schema should be auto crea
    // Be careful with this option and don't use this i
    // - otherwise you can lose production data.
    // This option is useful during debug and developme
    synchronize: true
}
```

- entities 는 나중에 생성할 엔티티 하나씩 넣어 줄 수도 있지만 아래처럼 작성하면 모든 엔티티를 다 포함하게 됩니다.
하나씩 작성하면 entities: [User, Board] User 엔티티와 Board 엔티티를 사용할 수 있게 지정...

```
{
...
"entities": ["src/bar/entities/**/*.ts"]
}
```

```
import {User} from "./payment/entity/User";
import {Post} from "./blog/entity/Post";

{
...
"entities": [User, Post]
}
```

3. 루트 Module에서 Import 합니다.

app.module.ts

```
@Module({
  imports: [
    TypeOrmModule.forRoot(typeORMConfig),
    BoardsModule
  ],
})
export class AppModule {}
```

forRoot안에 넣어준 설정(configuration)은
모든 Sub-Module 부수적인 모듈들에 다 적용이 됩니다.

게시물을 위한 엔티티(Entity) 생성하기

왜 Entity를 생성해야하나요?

원래 ORM 없이 데이터베이스 테이블을 생성할 때를 먼저 보겠습니다.

```
CREATE TABLE board (
  id     INTEGER AUTO_INCREMENT PRIMARY KEY.
  title  VARCHAR(255)  NOT NULL,
  decsription VARCHAR(255) NOT NULL
)
```

이런 식으로 테이블을 생성해줍니다. 하지만 TypeORM을 사용할 때는
데이터베이스 테이블로 변환 되는 Class이기 때문에 위에 처럼하지 않고

엔티티 생성 소스 코드

```
import { BaseEntity, Column, Entity, PrimaryG
import { BoardStatus } from "./boards.model";

@Entity()
export class Board extends BaseEntity {
    @PrimaryGeneratedColumn()
    id:number;

    @Column()
    title: string;

    @Column()
    description: string;

    @Column()
    status: BoardStatus;
}
```

@Entity()

@Entity()

- Entity () 데코레이터 클래스는 Board 클래스가 엔티티임을 나타내는 데 사용됩니다. CREATE TABLE board 부분입니다.

@PrimaryGeneratedColumn()

- PrimaryGeneratedColumn () 데코레이터 클래스는 id 열이 Board 엔터티의 기본 키 열임을 나타내는 데 사용됩니다.

@Column()

- Column () 데코레이터 클래스는 Board 엔터티의 title 및 description과 같은 다른 열을 나타내는 데 사용됩니다.

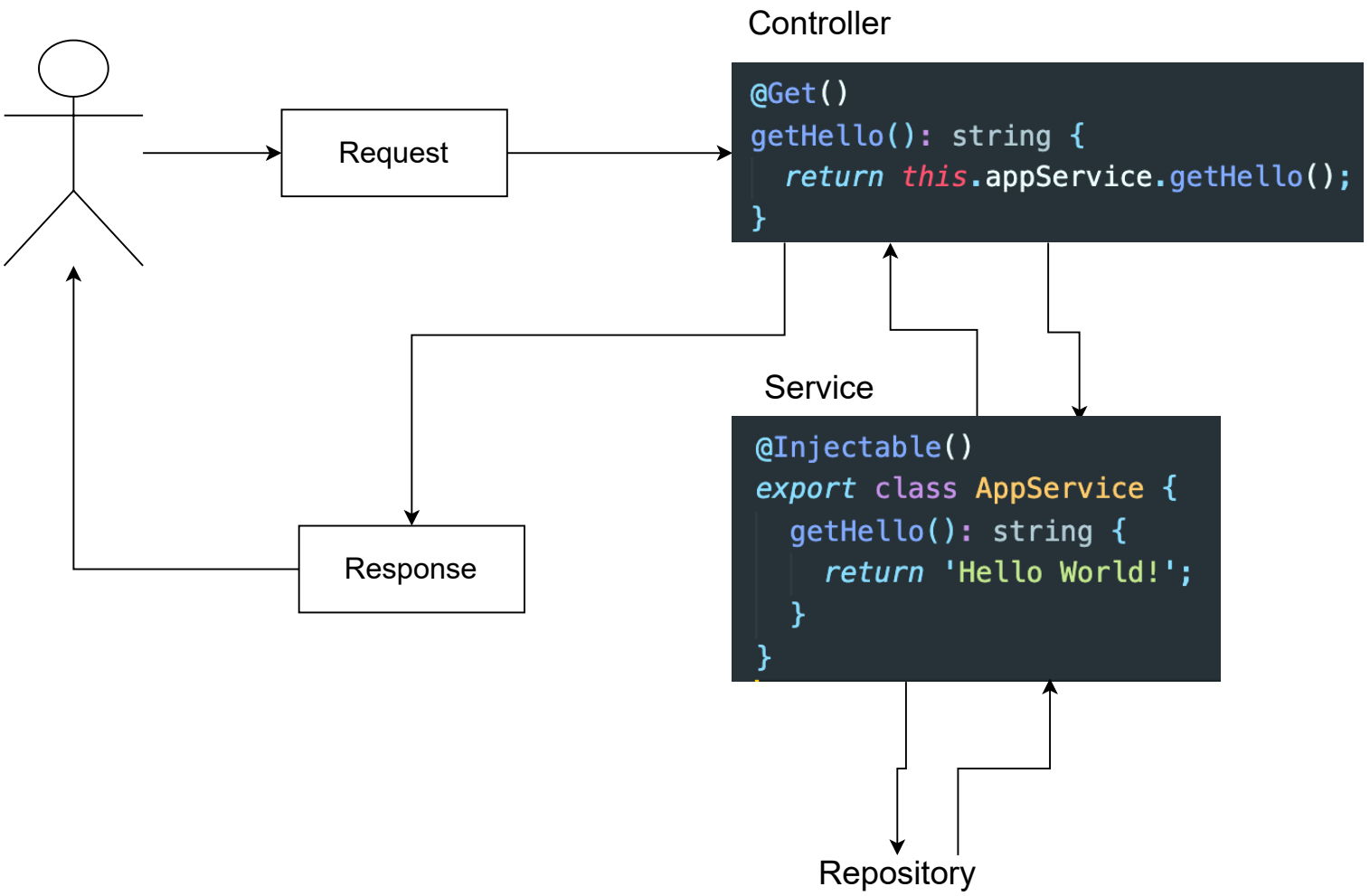
이제 Entity 클래스 Board가 생성되었습니다.
TypeORM은 데이터베이스의 Board 엔터티에 해당하는 테이블을 자동으로 생성하고 board로 이름을 지정합니다.

Repository 생성하기

Repository 란 무엇인가요 ?

리포지토리는 엔터티 개체와 함께 작동하며 엔터티 찾기, 삽입, 업데이트, 삭제 등을 처리합니다.

공식 문서 주소
http://typeorm.delightful.studio/classes/_repository_repository_.repository.html



데이터베이스에 관련 된 일은 서비스에서 하는게 아닌 Repository에서 해주시면 됩니다. 이것을 Repository Pattern 이라고도 부릅니다.

데이터베이스 관련 일 (INSERT, FIND, DELETE 등 등)

Repository 생성하기

1. 리포지토리 파일 생성하기

- board.repository.ts

2. 생성한 파일에 리포지토리를 위한 클래스 생성하기

- 생성 시 Repository 클래스를 Extends 해줍니다. (Find, Insert, Delete 등 엔티티를 컨트롤 해줄 수 있습니다.)

@EntityRepository()
- 클래스를 사용자 정의(CUSTOM) 저장소로 선언하는 데 사용됩니다. 사용자 지정 저장소는 일부 특정 엔터티를 관리하거나 일반 저장소 일 수 있습니다.

board.repositoty.ts

```
import { EntityRepository, Repository } from "typeorm";
import { Board } from "./board.entity";

@EntityRepository(Board)
export class BoardRepository extends Repository<Board> {

}
```

3. 생성한 Repository를 다른곳에서도 사용할 수 있기 위해서 (Injectable) board.module에서 import 해줍니다.

- board.module.ts

board.module.ts

```
@Module({
  imports: [
    TypeOrmModule.forFeature([BoardRepository]),
  ],
  controllers: [BoardsController],
  providers: [BoardsService]
})
```

```
export class BoardsModule { }
```