

## 이보경 코멘토 1주차 과제

### 1. 채팅 메시지 출력 화면

```
bokyung@bokyung-MacBookAir ~/Documents/1228_comento_muco main npm install
added 749 packages, and audited 750 packages in 42s

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

3 vulnerabilities (2 moderate, 1 critical)

To address all issues, run:
  npm audit fix

Run `npm audit` for details.
npm notice
npm notice New patch version of npm available! 10.2.4 -> 10.2.5
npm notice Changelog: https://github.com/npm/cli/releases/tag/v10.2.5
npm notice Run npm install -g npm@10.2.5 to update!
npm notice
bokyung@bokyung-MacBookAir ~/Documents/1228_comento_muco main npm run start:dev

> muco-chat@0.0.1 start:dev
> NODE_ENV=dev nest start --watch
[오후 1:47:00] Starting compilation in watch mode...

[오후 1:47:01] Found 0 errors. Watching for file changes.

[Nest] 51970 - 2023. 12. 30. 오후 1:47:02 LOG [NestFactory] Starting Nest application...
[Nest] 51970 - 2023. 12. 30. 오후 1:47:02 LOG [InstanceLoader] DataModule dependencies initialized +8ms
[Nest] 51970 - 2023. 12. 30. 오후 1:47:02 LOG [InstanceLoader] TypeOrmModule dependencies initialized +0ms
[Nest] 51970 - 2023. 12. 30. 오후 1:47:02 LOG [InstanceLoader] ConfigHostModule dependencies initialized +0ms
[Nest] 51970 - 2023. 12. 30. 오후 1:47:02 LOG [InstanceLoader] ConfigModule dependencies initialized +0ms
[Nest] 51970 - 2023. 12. 30. 오후 1:47:02 LOG [InstanceLoader] ConfigModule dependencies initialized +0ms
[Nest] 51970 - 2023. 12. 30. 오후 1:47:06 LOG [InstanceLoader] TypeOrmCoreModule dependencies initialized +4341ms
[Nest] 51970 - 2023. 12. 30. 오후 1:47:06 LOG [InstanceLoader] TypeOrmModule dependencies initialized +0ms
[Nest] 51970 - 2023. 12. 30. 오후 1:47:06 LOG [InstanceLoader] TypeOrmModule dependencies initialized +1ms
[Nest] 51970 - 2023. 12. 30. 오후 1:47:06 LOG [InstanceLoader] TypeOrmModule dependencies initialized +0ms
```

채팅

공지가 없습니다

LE SSERAFIM (르세라핌): 3000번 port!!! | liked: false count: 0

LE SSERAFIM (르세라핌): 3001번 port!!! | liked: false count: 0

IVE (아이브): 1228 | liked: false count: 0

AKMU (악뮤): 1228 | liked: false count: 0

IVE (아이브): ggg | liked: false count: 0

AKMU (악뮤): zzzz | liked: false count: 0

IVE (아이브): 엔티어라본 하이요 | liked: false count: 0

AKMU (악뮤): zzzzzz | liked: false count: 0

LE SSERAFIM (르세라핌): hi | liked: false count: 0

LE SSERAFIM (르세라핌): hi | liked: false count: 0

메시지:  전송

## 2. NestJS

### 1) NestJS 소개

NestJS는 효율적이고 확장 가능한 서버 사이드 애플리케이션을 구축하기 위한 프레임워크입니다. TypeScript로 작성되었으며, 주요 특징 중 하나는 의존성 주입 시스템(DI)이라는 것입니다. 이를 통해 강력한 모듈성을 갖게 되고, 간결하고 유지관리가 쉬운 코드를 작성할 수 있습니다.

모듈은 관련된 컨트롤러와 provider(서비스 등)의 그룹으로 구성되며, 각 모듈은 애플리케이션의 특정 부분을 맡게 됩니다. 이를 통해 유지보수성과 확장성을 향상시킵니다.

또한, OOP(객체 지향 프로그래밍), FP(기능 프로그래밍) 및 FRP(기능 반응형 프로그래밍)의 요소를 결합합니다.

NestJS는 아래와 같은 상황에 적합하다고 할 수 있습니다.

- ① 대규모 및 복잡한 백엔드 시스템
- ② 실시간 기능이 필요한 애플리케이션 (웹 소켓 지원)
- ③ 마이크로서비스 아키텍처
- ④ TypeScript를 선호하는 경우
- ⑤ 테스트 주도 개발(TDD)을 선호하는 경우

### 2) 의존성 주입 (DI)

기본적인 의미는 외부에서 서비스를 주입(제공)하는 것입니다. 다시 말해, 클래스 간 의존성을 클래스 외부에서 주입하는 것으로 객체가 의존하는 또 다른 객체를 외부에서 선언하고 이를 주입받아 사용하는 것이라고 볼 수 있습니다.

@Injectable() 데코레이터를 통해 의존성 주입의 대상임을 알려줄 수 있습니다. (provider)

### 3) Provider

provider는 기본적으로 클래스, 값, 팩토리 등을 생성할 수 있는 공급원입니다. 이러한 provider들은 의존성 주입 시스템을 통해 다른 곳에서 사용할 수 있게 됩니다. 이를 통해 코드를 재사용하고, 테스트를 용이하게 하며, 응용 프로그램의 부분들을 느슨하게 결합할 수 있습니다.

provider는 클래스를 제공하는 가장 일반적인 방법입니다.

### 4) Controller

컨트롤러는 들어오는 요청을 처리하고 응답을 반환하는 역할을 수행합니다.

클라이언트로부터 들어온 HTTP 요청들은 라우팅 메커니즘에 의해 알맞은 컨트롤러로 분배됩니다.

NestJS에서는 @Controller, @Get, @Post 등 데코레이터를 이용하여 컨트롤러를 구성합니다.

각각의 컨트롤러는 하나 이상의 라우트를 가질 수 있으며, 라우트는 클라이언트가 어떠한 작업을 요청했는지를 결정합니다.

HTTP를 통해 들어오는 요청을 처리하고, 적절한 서비스를 호출하여 비즈니스 로직을 수행하며 그 결과를 클라이언트에 반환합니다.

@Controller 데코레이터에 경로를 지정하면 사용자가 해당 경로로 접근할 수 있습니다.

```

@Controller('users')
export class UsersController {
  constructor(private readonly userService: UsersService) {}

  @Post()
  create(@Body() createUserDto: CreateUserDto) {
    return this.userService.create(createUserDto);
  }

  @Get()
  findAll() {
    return this.userService.findAll();
  }

  @Get(':id')
  findOne(@Param('id') id: string) {
    return this.userService.findOne(+id);
  }

  @Patch(':id')
  update(@Param('id') id: string, @Body() updateUserDto: UpdateUserDto) {
    return this.userService.update(+id, updateUserDto);
  }

  @Delete(':id')
  remove(@Param('id') id: string) {
    return this.userService.remove(+id);
  }
}

```

이와 같이 기본적인 CRUD 컨트롤러를 구성할 수 있습니다.

## 5) Service

서비스는 provider의 한 종류로, 비즈니스 로직을 담당합니다. 컨트롤러와 달리 서비스는 실제로 데이터를 처리하며, 데이터베이스와의 상호작용, 복잡한 알고리즘 실행, 서드파티 리소스 호출 등의 작업을 담당합니다.

서비스는 재사용 가능하며, 컨트롤러와 분리되어 있어 코드의 유지보수성을 향상시킵니다.

```
import { Injectable } from '@nestjs/common';
import { Cat } from '../interfaces/cat.interface';

@Injectable()
export class CatsService {
  private readonly cats: Cat[] = [];

  create(cat: Cat) {
    this.cats.push(cat);
  }

  findAll(): Cat[] {
    return this.cats;
  }
}
```

위와 같이 서비스를 구성할 수 있습니다. 위 서비스는 데이터 저장 및 검색을 수행하며, CatsController에서 사용하도록 구성되었습니다.

## 6) TypeORM

ORM은 객체와 관계형 데이터베이스의 데이터를 자동으로 연결해주는 작업을 의미합니다.

TypeORM은 TypeScript와 JavaScript와 함께 사용할 수 있습니다. MySQL, MariaDB, PostgreSQL, SQLite, SQL Server 등 다양한 데이터베이스를 지원합니다.

NestJS에서 TypeORM을 사용하기 위해 @nestjs/typeorm 패키지와 데이터베이스 드라이버를 설치해야 합니다. 그 다음 NestJS 모듈에서 TypeORM 모듈을 import하고, 설정을 통해 데이터베이스에 연결합니다. 이후 서비스에서 레포지토리를 주입받아 사용합니다.

NestJS에서 TypeORM은 데이터베이스 엔티티와 상호작용하는 데 필요한 도구이며, 아래와 같은 특징을 가집니다.

- ① **모듈성**: NestJS는 모듈 기반 구조를 가지고 있으며, TypeORM은 @nestjs/typeorm 모듈을 통해 쉽게 통합됩니다. 이를 통해 애플리케이션의 다른 부분과의 결합도를 낮추고, 코드의 단위 테스트가 용이해집니다.
- ② **의존성 주입**: 서비스에 TypeORM 레포지토리를 주입하여 DB 연산을 캡슐화하고 재사용 가능한 서비스를 만들 수 있습니다.
- ③ **데코레이터 사용**: TypeORM은 엔티티를 정의할 때 데코레이터를 사용하는데, NestJS도 이러한 패턴을 채택하고 있어 일관성을 유지할 수 있습니다.
- ④ **트랜잭션 관리**: 데코레이터를 사용해 쉽게 트랜잭션을 관리할 수 있고, 이를 통해 복잡한 비즈니스 로직에서 데이터의 일관성을 유지할 수 있습니다.
- ⑤ **쿼리 빌더**: TypeORM의 쿼리 빌더를 사용하면 복잡한 쿼리도 쉽게 작성할 수 있으며, 코드의 가독성을 높일 수 있습니다.

즉, NestJS와 TypeORM의 조합은 Node.js 환경에서 타입 안전성, 확장성 및 유지보수성을 갖춘 서버 사이드 애플리케이션을 구축하는 데 매우 유용합니다.

## 7) 테스트 코드 작성

### ① 단위 테스트

#### A. 테스트 환경 설정

- Jest 설정하고 필요한 모듈을 import합니다. (Jest는 기본 설정)

#### B. 테스트 모듈 생성

- Test.createTestingModule 메서드를 이용하여 테스트할 클래스와 필요한 provider를 포함하는 테스트 모듈을 생성합니다.

#### C. 의존성 주입

- DI 시스템을 사용하여 테스트 인스턴스를 가져오고, 필요한 경우 mock 객체를 주입합니다.
- mock 객체: 실제 객체를 테스트 환경에서 대체하기 위해 사용하는 객체. 주로 단위 테스트에서 실제 구현 대신 테스트를 위한 간단한 구현을 제공하거나, 아직 개발되지 않은 기능을 대신하거나, 실제로 실행하기에는 비용이나 시간이 많이 드는 작업을 대신할 때 사용

#### D. 테스트 케이스 작성

- it 또는 test 함수를 사용하여 개별 테스트 케이스를 작성합니다.
- 각 테스트 케이스에서는 expect 함수를 사용하여 기대하는 결과를 확인합니다.

```
import { Test,TestingModule } from '@nestjs/testing';
import { YourService } from './your.service';

describe('YourService', () => {
  let service: YourService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [YourService],
    }).compile();

    service = module.get<YourService>(YourService);
  });

  it('should be defined', () => {
    expect(service).toBeDefined();
  });

  it('should do something', () => {
    expect(service.someMethod()).toEqual('expected result');
  });
});
```

## ② E2E 테스트 (End-to-End)

E2E 테스트는 사용자의 관점에서 애플리케이션의 흐름을 테스트합니다.

Supertest를 사용하여 HTTP 요청을 보내고 응답을 검증합니다.

```
import { Test, TestingModule } from '@nestjs/testing';
import * as request from 'supertest';
import { AppModule } from './../../src/app.module';

describe('AppController (e2e)', () => {
  let app;

  beforeEach(async () => {
    const moduleFixture: TestingModule = await Test.createTestingModule({
      imports: [AppModule],
    }).compile();

    app = moduleFixture.createNestApplication();
    await app.init();
  });

  it('/ (GET)', () => {
    return request(app.getHttpServer())
      .get('/')
      .expect(200)
      .expect('Hello World!');
  });

  afterAll(async () => {
    await app.close();
  });
});
```

### 3. TypeORM 사용법

#### 1) TypeORM, DB 드라이버 설치

```
npm install --save @nestjs/typeorm typeorm
npm install --save pg
```

#### 2) 모듈 설정

@Module 데코레이터 내에 TypeORM 모듈을 설정합니다.

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';

@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: 'localhost',
      port: 5432,
      username: 'bokyung',
      password: '1234',
      database: 'test',
      entities: [__dirname + '/../**/*.entity{.ts,.js}'],
      synchronize: false, // production
    }),
  ],
})
export class AppModule {}
```

#### 3) 엔티티 생성

데이터베이스 테이블과 매핑될 엔티티 클래스를 생성합니다.

```
import { Entity, PrimaryGeneratedColumn, Column } from 'typeorm';

@Entity()
export class User {
  @PrimaryGeneratedColumn()
  id: number;

  @Column()
  firstName: string;

  @Column()
  lastName: string;

  @Column()
  isActive: boolean;
}
```

User라는 이름의 엔티티를 정의하고, 각 컬럼에 대한 타입을 지정했습니다.

#### 4) 레포지토리 사용

엔티티의 레포지토리를 주입받아 DB 연산을 수행합니다.

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Repository } from 'typeorm';
import { User } from '../data/entity/user.entity';

@Injectable()
export class UserService {
  constructor(
    @InjectRepository(User)
    private usersRepository: Repository<User>,
  ) {}

  findAll(): Promise<User[]> {
    return this.usersRepository.find();
  }

  // ... 추가적인 메서드들
}
```

User 엔티티의 레포지토리를 주입받아 findAll 메서드를 통해 모든 유저를 조회할 수 있게 합니다.

#### 5) 서비스 및 컨트롤러 연결

생성한 서비스를 컨트롤러에 주입하고, 컨트롤러에서 HTTP 요청을 처리하도록 합니다.

```
import { Controller, Get } from '@nestjs/common';
import { UserService } from '../user.service';
import { User } from '../data/entity/user.entity';

@Controller('users')
export class UserController {
  constructor(private readonly userService: UserService) {}

  @Get()
  findAll(): Promise<User[]> {
    return this.userService.findAll();
  }

  // 추가적인 라우트 핸들러들
}
```

GET 요청을 /users 경로로 받아 UserService의 findAll 메서드를 호출하여 결과를 반환합니다.



#### 4. Postgres 설치

```
bokyung@bokyung-MacBookAir ~$ docker run -d -p 5432:5432 -e POSTGRES_PASSWORD="dlqhrud-1124" --name bokyung postgres
2147fe967f2e903d894928267ecf958503e104f369200450a9571fbdec71e6d7
bokyung@bokyung-MacBookAir ~$ docker run -d -p 5432:5432 -e POSTGRES_PASSWORD="dlqhrud-1124" --name PostgreSQL01 postgres
764e8613d9be164527aadbeb368febe82728559cd5716ea8529b2fb4a413fe84
bokyung@bokyung-MacBookAir ~$ docker exec --user root -it PostgreSQL01 sh
# psql -U postgres
psql (16.1 (Debian 16.1-1.pgdg120+1))
Type "help" for help.

postgres=# select datname from pg_database;
 datname
-----
 postgres
 template1
 template0
(3 rows)
```

docker를 이용하여 postgres를 실행했습니다.