[코멘토] 채팅 서비스 마이그레이션

1. 개발 환경 설정: NestJS 환경 설정

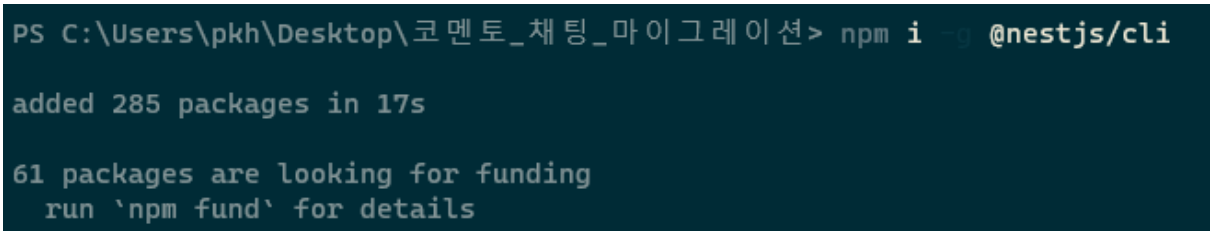
Node.js 설치하기

https://nodejs.org/en

NestJS CLI 설치

cmd에서 명령으로 cli를 설치한다

```
$ npmi -g @nestjs/cli
```



프로젝트 생성 (직접 생성 시)

```
$ nest new project-name
```

프로젝트 다운로드시

- 깃허브에서 원하는 링크 다운 받기

```
git clone https://github.com/moo-co/240108_moocochat.git
```

```
PS C:\Users\pkh\Desktop\코멘토_채팅_마이그레이션> git clone https://github.com/moo-co/240108_moocochat.git
Cloning into '240108_moocochat'...
remote: Enumerating objects: 255, done.
remote: Counting objects: 100% (255/255), done.
remote: Compressing objects: 100% (99/99), done.
Receiving objects:  70% (179/255)used 253 (delta 140), pack-reused 0
Receiving objects: 100% (255/255), 201.58 KiB | 6.11 MiB/s, done.
Resolving deltas: 100% (141/141), done.
```

프로젝트 실행

- 해당 레포지토리로 이동 및 npm 파일 다운로드

```
cd .\240108_moocochat\
npm install
```

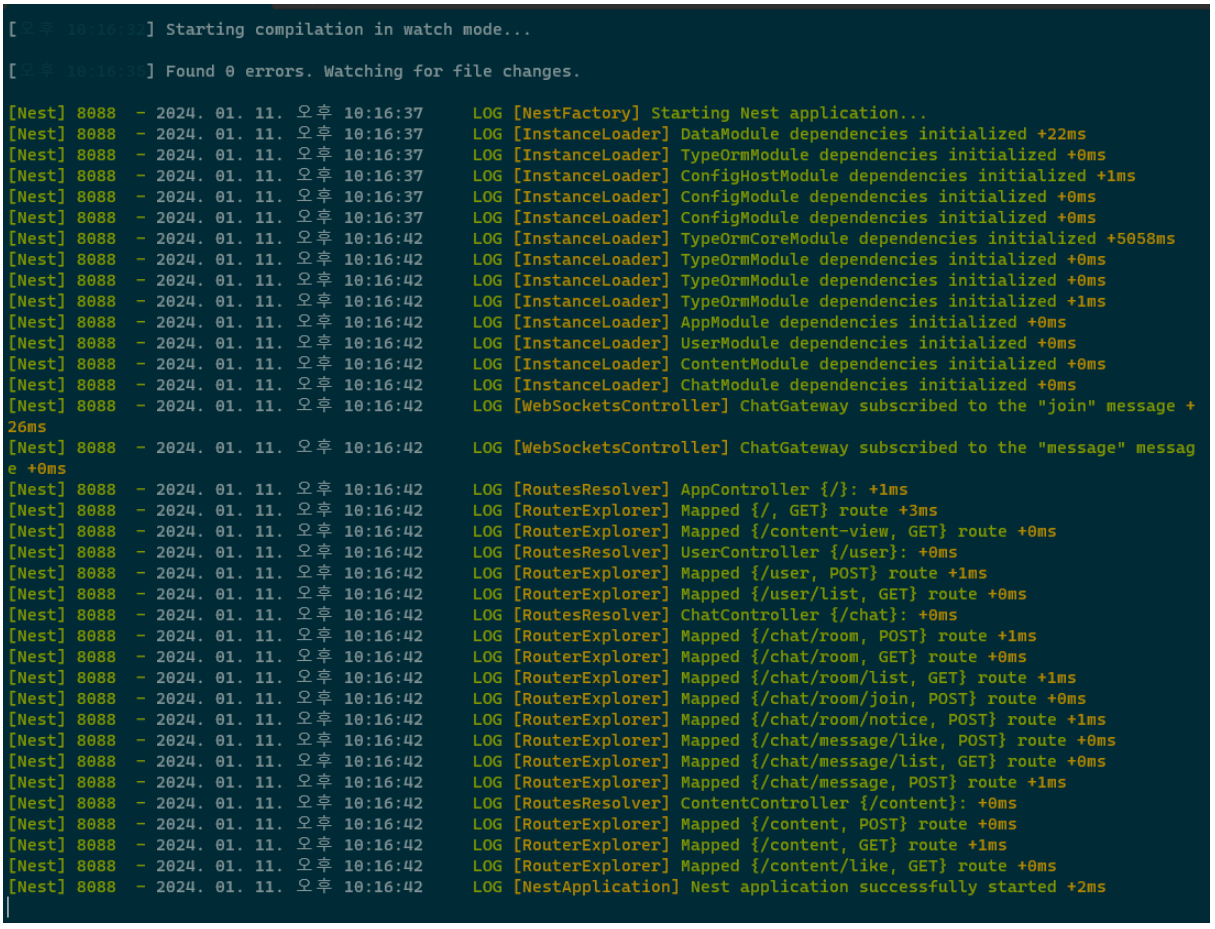
```
PS C:\Users\pkh\Desktop\코멘토_채팅_마이그레이션> cd .\240108_moocochat\
PS C:\Users\pkh\Desktop\코멘토_채팅_마이그레이션\240108_moocochat> npm i

added 748 packages, and audited 749 packages in 29s

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.
```

- 실행 명령으로 실행하기

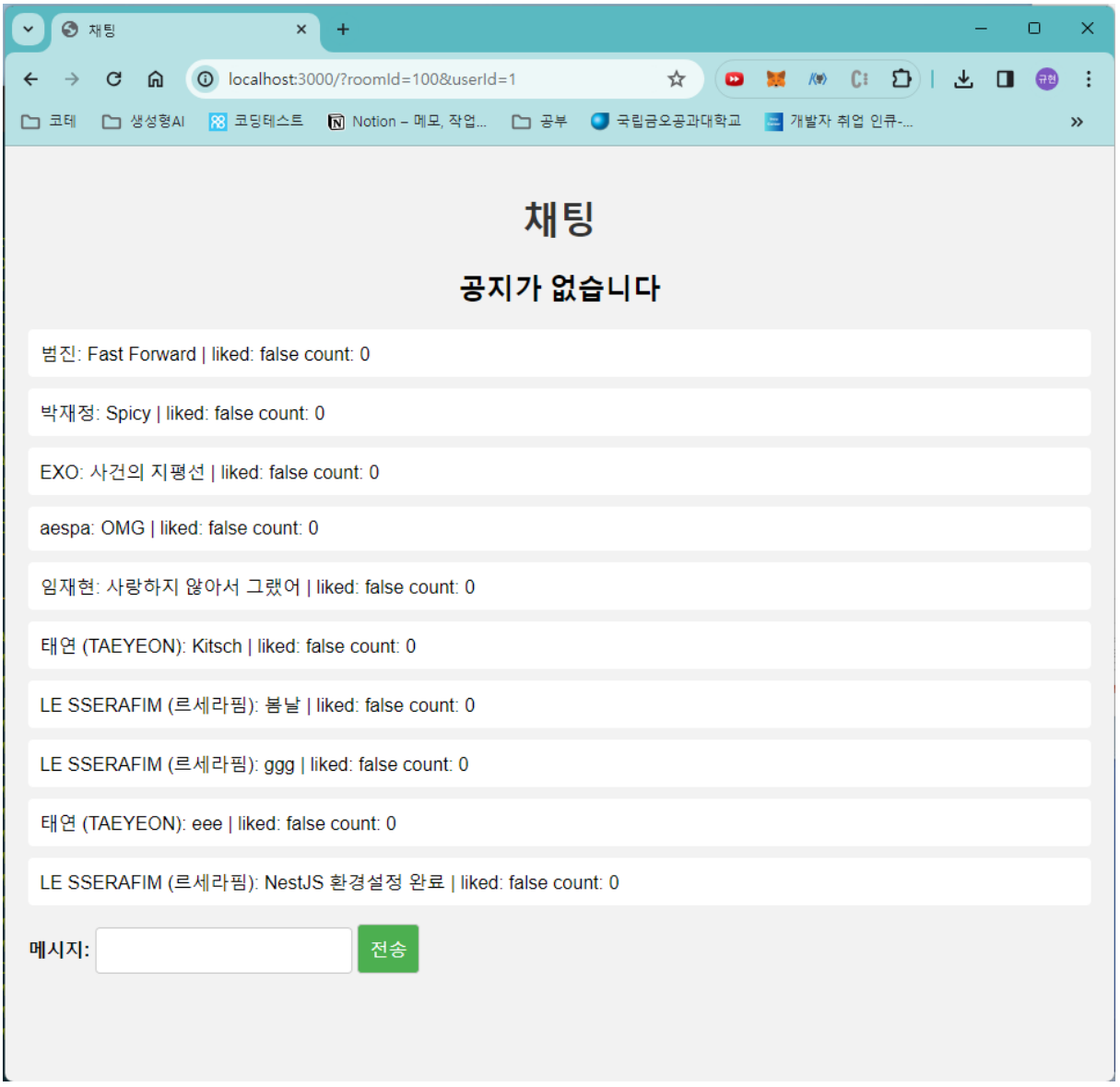
```
npm run start
```

- 무코챗은 아래 명령으로 실행하기

```
npm run start:dev
```



```
[오후 10:16:32] Starting compilation in watch mode...

[오후 10:16:35] Found 0 errors. Watching for file changes.

[Nest] 8088  - 2024. 01. 11. 오후 10:16:37     LOG [NestFactory] Starting Nest application...
[Nest] 8088  - 2024. 01. 11. 오후 10:16:37     LOG [InstanceLoader] DataModule dependencies initialized +22ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:37     LOG [InstanceLoader] TypeOrmModule dependencies initialized +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:37     LOG [InstanceLoader] ConfigHostModule dependencies initialized +1ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:37     LOG [InstanceLoader] ConfigModule dependencies initialized +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:37     LOG [InstanceLoader] ConfigModule dependencies initialized +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [InstanceLoader] TypeOrmCoreModule dependencies initialized +5058ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [InstanceLoader] TypeOrmModule dependencies initialized +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [InstanceLoader] TypeOrmModule dependencies initialized +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [InstanceLoader] TypeOrmModule dependencies initialized +1ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [InstanceLoader] AppModule dependencies initialized +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [InstanceLoader] UserModule dependencies initialized +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [InstanceLoader] ContentModule dependencies initialized +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [InstanceLoader] ChatModule dependencies initialized +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [WebSocketsController] ChatGateway subscribed to the "join" message +
26ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [WebSocketsController] ChatGateway subscribed to the "message" messag
e +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RoutesResolver] AppController {/}: +1ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/, GET} route +3ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/content-view, GET} route +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RoutesResolver] UserController {/user}: +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/user, POST} route +1ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/user/list, GET} route +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RoutesResolver] ChatController {/chat}: +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/chat/room, POST} route +1ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/chat/room, GET} route +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/chat/room/list, GET} route +1ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/chat/room/join, POST} route +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/chat/room/notice, POST} route +1ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/chat/message/like, POST} route +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/chat/message/list, GET} route +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/chat/message, POST} route +1ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RoutesResolver] ContentController {/content}: +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/content, POST} route +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/content, GET} route +1ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [RouterExplorer] Mapped {/content/like, GET} route +0ms
[Nest] 8088  - 2024. 01. 11. 오후 10:16:42     LOG [NestApplication] Nest application successfully started +2ms
```



2. NestJS에 대해 알아보자

프로젝트 구조

- dist: 타입스크립트가 자바스크립트로 컴파일되는 결과물이 담기는 곳
- node_modules : npm을 통해 설치된 모듈에 관련된 내용들이 담기는 곳
- src : 작업 해야하는 곳
- test : 테스트에 관한 내용들이 담기는 곳

기본 로직

NestJS의 어플리케이션 동작 순서는 `AppModule -> AppController -> AppService` 의 흐름으로 진행됩니다. 즉, **MVC** 패턴을 따른다고 생각하면 됩니다.

NestJS의 구성 요소

들어가기 전

다음에 나오는 `모듈, 컨트롤러, 서비스`를 알기 전에 알아둬야 하는 요소들이 있다.

- Provider : 공급자란 애플리케이션에서 사용되는 객체를 생성하고 제공하는 역할

모듈 (Module)

- 관련된 컨트롤러, 서비스 및 기타 provider 들을 하나로 묶는 역할
- 계층적인 구조로 구성
- 의존성 주입 및 코드의 모듈화를 통해 애플리케이션의 `확장성` 과 `유지 보수성`을 향상 시킴
- 생성 CLI : `nest g module '모듈명'`

```
// cats.module.ts
import { Module } from '@nestjs/common';
import { CatsController } from './cats.controller';
import { CatsService } from './cats.service';

@Module({
  controllers: [CatsController],
  providers: [CatsService],
})
export class CatsModule {}
```

컨트롤러 (Controller)

- 클라이언트의 요청을 처리
- 서비스로부터 데이터를 받는 역할

- 라우팅의 역할을 수행
- 생성 CLI : `nest g controller '컨트롤러명'`

```
// cats.constoller.ts
import { Controller } from '@nestjs/common';
import { CatsService } from './cats.service';

@Controller('cats')
export class CatsController {
  constructor(private readonly catsService: CatsService) {}

  @Get()
  async findAllCats(): Promise<any[]> {
    this.catsService.findAllCats();
  }
}
```

서비스 (Service)

- 비즈니스 로직을 담당
- 컨트롤러로부터 전달받은 요청을 처리
- DB나 외부 API와의 상호작용을 수행
- 생성 CLI : `nest g service '서비스명'`

```
// cats.service.ts
import { Injectable } from '@nestjs/common'

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

  async findAllCats(): Promise<any[]> {
    return this.cats;
  }
}
```

모델 (Model)

- 데이터 구조와 상호작용을 관리하는 부분
  - TypeORM을 통해 model은 Entity로 대체될 수 있음
- Class를 이용하거나 Interface를 이용
  - class : 변수의 타입, 인스턴스 생성 등
  - interface : 변수의 타입만을 체크

```
// ./src/boards/boards.model.ts
export interface Board {
    id: string;
    title: string;
    description: string;
    status: BoardStatus;
}

export enum BoardStatus {
    PUBLIC = 'PUBLIC',
    PRIVATE = 'PRIVATE',
}
```

DTO (Data Transfer Object)

- 계층간 데이터 교환을 위한 객체
- DB에서 데이터를 얻어 Service나 Controller 등으로 보낼 때 사용하는 객체
- DTO는 데이터가 네트워크를 통해 전송되는 방법을 정의하는 객체
- Interface나 class를 이용해서 정의

```
// ./src/boards/create-board.dto.ts
import { IsNotEmpty } from 'class-validator';

export class CreateBoardDto {
    @IsNotEmpty
    title: string;

    @IsNotEmpty
```

```
    description: string;
}
```

DTO를 쓰는 이유

1) 데이터 유효성을 체크하는데 효율적

2) 유지보수과정에서 다수의 프로퍼티를 관리 가능

3) 타입스크립트의 타입으로도 사용 가능

PIPE

사용하기 전에 `npm install class-validator class-transformer --save` 를 통해 설치해야 함

- 파이프는 **data transformation**과 **data validation**을 위해 사용
  - **Data Transformation**
    - *입력 데이터를 원하는 형식으로 변환*
    - Ex) 만약 숫자를 받길 원할 시 데이터가 문자열 형식으로 온다면 파이프에서 자동으로 숫자로 변환
  - **Data Validation**
    - *입력 데이터를 평가 후 전달 or 에러 처리*
- 파이프는 컨트롤러 경로 처리기에 의해 처리되는 인수로 작동
- Nest는 메소드가 호출되기 직전에 파이프를 삽입하고 파이프는 메소드로 향하는 인수를 수신 후 작동
- **PIPE 사용법**

  1) Handler-level Pipes
  - 라우터 핸들러 레벨에 직접 @UsePipes() 데코레이터를 이용하여 모든 파라미터에 적용

  2) Parameter-level Pipes
  - 특정한 파라미터에 적용되는 파이프

  3) Global-level Pipes
  - 애플리케이션 레벨의 파이프로 클라이언트에서 들어오는 모든 요청에 적용

3. TypeORM

TypeORM이란?

TypeORM은 TypeScript에서 관계형 데이터베이스와 작업을 수행하는 데 사용되는 ORM(Object-Relational Mapping) 프레임워크입니다.

타입ORM 설치 및 설정

타입ORM을 사용하기 위해 다음과 같이 모듈을 설치하고 설정합니다.

모듈 설치

```
npm i mysql2 typeorm @nestjs/typeorm --save
```

TypeORM 설정

타입ORM의 설정은 다음과 같이 typeorm.config.ts 파일을 생성하여 데이터베이스의 설정 정보를 담을 수 있습니다.

```
// ./src/configs/typeorm.config.ts
import { TypeOrmModuleOptions } from '@nestjs/common';

export const typeORMConfig: TypeOrmModuleOptions = {
  type: 'mysql',
  host: 'localhost',
  port: 5432,
  username: 'yourusername',
  password: 'yourpassword',
  database: 'yourdatabase',
  entities: [__dirname + '/../**/*.entity.{js,ts}'],
};
```

TypeORM 적용

```
// ./src/app.module.ts
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { BoardModule } from './board/board.module';
import { typeORMConfig } from './configs/typeorm.config';

@Module({
  imports: [TypeOrmModule.forRoot(typeORMConfig), BoardModu
le],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

참고

- NestJS로 배우는 백엔드 프로그래밍 (위키독스)
- https://velog.io/@jyunimyon/Nest-JS-Nest-JS-1-설치부터-예제까지
- https://velog.io/@chaeheetae/nestjs의-기본개념#3-nestjs의-주요-요소
- https://develop-const.tistory.com/7#:~:text=공급자란%3F,를 공급자로 등록합니다.
- https://velog.io/@kimkevin90/NestJs#nestjs-구성-요소