

08. 요구사항-1 회원가입

- (일반회원) 사용자는 이메일과 비밀번호로 회원가입할 수 있어야 합니다.
- 이메일은 중복되지 않아야 합니다.
- 비밀번호는 안전한 방식으로 저장되어야 합니다.

1. 현 상황 정리

auth.service.ts

```
import { Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import * as bcrypt from 'bcrypt';

@Injectable()
export class AuthService {
  constructor(private jwtService: JwtService) {}

  // 사용자 검증 로직
  async validateUser(username: string, pass: string): Promise<any> {
    // 여기에 사용자 검증 로직을 구현합니다.
    // 예: 데이터베이스에서 사용자 정보를 조회하고 비밀번호를 비교합니다.
  }

  // 로그인 로직
  async login(user: any) {
    const payload = { username: user.username, sub: user.userId };
    return {
      access_token: this.jwtService.sign(payload),
    };
  }

  // 비밀번호 해시 생성
  async hashPassword(password: string): Promise<string> {
    const salt = await bcrypt.genSalt();
    return bcrypt.hash(password, salt);
  }
}
```

```
// 저장된 해시와 비밀번호 비교
async comparePasswords(password: string, storedPasswordHash: string):
Promise<boolean> {
    return bcrypt.compare(password, storedPasswordHash);
}
}
```

app.controller.ts

```
// app.controller.ts
import { Controller, Post, Body, Get } from '@nestjs/common';
import { AuthService } from './auth.service';

@Controller('users')
export class AppController {
    constructor(private readonly authService: AuthService) {}

    @Post('register')
    async register(@Body() userData: RegisterDto) {
        // 회원가입 로직
    }

    @Post('login')
    async login(@Body() loginData: LoginDto) {
        // 로그인 로직
    }

    @Get()
    async findAll() {
        // 모든 사용자 조회
    }
}
```

- RegisterDto 구현
- 요구사항에 맞게 회원가입 로직 작성

2. RegisterDto 구현

```
// src/dto/register.dto.ts
import { IsEmail, IsNotEmpty, IsString, MinLength } from 'class-validator';

export class RegisterDto {
  @IsEmail()
  @IsNotEmpty()
  email: string;

  @IsString()
  @MinLength(6)
  @IsNotEmpty()
  password: string;
}
```

2-1. 필요 모듈 설치

```
npm install class-validator
npm install class-transformer
```

2-2. RegisterDto 등록

app.controller.ts

```
import { RegisterDto } from '../dto/register.dto';
```

3. AppController 구현

```
// app.controller.ts
import { Controller, Post, Body } from '@nestjs/common';
import { AuthService } from '../auth.service';
import { UserService } from '../user.service';
import { RegisterDto } from '../dto/register.dto';

@Controller('users')
export class AppController {
  constructor(
    private readonly authService: AuthService,
    private readonly userService: UserService
  ) {}

  @Post()
  register(@Body() registerDto: RegisterDto) {
    // ...
  }
}
```

```

) {}

@Post('register')
async register(@Body() registerDto: RegisterDto) {
  // 이메일 중복 검사
  const existingUser = await this.userService.findByEmail(registerDto.email);
  if (existingUser) {
    throw new Error('이미 존재하는 이메일입니다.');
```

```

  // 비밀번호 해싱
```

```

  const hashedPassword = await
this.authService.hashPassword(registerDto.password);
```

```

  // 사용자 정보 저장
```

```

  const user = await this.userService.createUser({
    ...registerDto,
    password: hashedPassword,
  });
```

```

  // 응답 반환 (비밀번호 정보는 제외)
```

```

  return { id: user.id, email: user.email };
}
```

```

// 기타 메서드...
```

```

}
```

4. save에 return 삭제

- return 타입이 자꾸 안맞아서 save에 return 을 void로 하기로 결정하였다.

5. user.entity.ts 수정

- username을 삭제하기로 하였다.(이메일로 가입하는데 unique 제약이 걸려있어 삭제하는게 나을듯하다.)
- role을 enum 타입으로 지정하여 강제성을 띄게 바꾸었다.

6. 수정된 전체 코드

- user.entity.ts

```

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

// UserRole enum 정의
export enum UserRole {
    MEMBER = 'MEMBER',
    ADMIN = 'ADMIN',
}

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

    @Column({ unique: true })
    email: string;

    @Column()
    password: string;

    @Column({
        type: 'enum',
        enum: UserRole,
        default: UserRole.MEMBER
    })
    role: UserRole;
}

```

- user.service.ts

```

// user.service.ts
import { Injectable } from '@nestjs/common';
import { Repository } from 'typeorm';
import { User } from '../entities/user.entity';
import { InjectRepository } from '@nestjs/typeorm';

@Injectable()
export class UserService {
    constructor(

```

```

    @InjectRepository(User)
    private readonly userRepository: Repository<User>,
  ) {}

  async findByEmail(email: string): Promise<User | undefined> {
    return this.userRepository.findOne({ where: { email } });
  }

  async createUser(userData: any): Promise<void> {
    const user = this.userRepository.create(userData);
    await this.userRepository.save(user);
  }

  // 기타 메서드...
}

```

- app.controller.ts

```

// app.controller.ts
import { Controller, Post, Body, Get } from '@nestjs/common';
import { AuthService } from '../auth.service';
import { UserService } from '../user.service';
import { RegisterDto } from '../dto/register.dto';

@Controller('users')
export class AppController {
  constructor(
    private readonly authService: AuthService,
    private readonly userService: UserService
  ) {}

  @Post('register')
  async register(@Body() registerDto: RegisterDto) {
    // 이메일 중복 검사
    const existingUser = await this.userService.findByEmail(registerDto.email);
    if (existingUser) {

```

```

        throw new Error('이미 존재하는 이메일입니다.');
```

```

    }

    // 비밀번호 해싱
    const hashedPassword = await
this.authService.hashPassword(registerDto.password);

    // 사용자 정보 저장
    await this.userService.createUser({
        ...registerDto,
        password: hashedPassword,
    });

    // 응답 반환 (비밀번호 정보는 제외)
    return { email: registerDto.email };
}

// @Post('login')
// async login(@Body() loginData: LoginDto) {
//     // 로그인 로직
// }

@Get()
async findAll() {
    // 모든 사용자 조회
    return 'hello world';
}
}

```

7. 결과 확인

- postman으로 api 요청을 보낸 결과 다음과 같이 정상적으로 api 호출이 되는 것을 확인할 수 있었다.

HTTP http://localhost:3000/users/register

POST http://localhost:3000/users/register

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   "email": "example@example.com",
3   "password": "yourpassword"
4 }
5

```

Body Cookies Headers (7) Test Results Status: 201 Created Time: 116 ms Size: 271 B Save as example

Pretty Raw Preview Visualize JSON

```

1 {
2   "email": "example@example.com"
3 }

```

- mysql에도 값 잘 들어간다

```

mysql> SELECT * FROM opstest.user;
+----+-----+-----+-----+
| id | email                | password                                     |
| role |
+----+-----+-----+-----+
| 1 | example@example.com | $2b$10$J11c5tJv3UW0ZAPvXC2Xz.1JOS1.7tv4g3WjvNKUFeoQT9hiSVgAW | MEMBER |
+----+-----+-----+-----+
1 row in set (0.01 sec)

mysql>

```

8. 요구사항 체크


- (일반회원) 사용자는 이메일과 비밀번호로 회원가입할 수 있어야 합니다. ✓
- 이메일은 중복되지 않아야 합니다. ✓

- 같은 이메일을 넣었을 때 다음과 같이 정상적으로 값을 받는 것을 볼 수 있다.

```
[Nest] 24708 - 2024. 01. 24. 오후 6:21:39 ERROR [ExceptionHandler] 이미 존재하는 이메일입니다.
```

```
Error: 이미 존재하는 이메일입니다.
```

```
    at AppController.register (C:\gits\ops-test\src\app.controller.ts:19:13)
    at processTicksAndRejections (node:internal/process/task_queues:95:5)
    at C:\gits\ops-test\node_modules\@nestjs\core\router\router-execution-
context.js:46:28
    at C:\gits\ops-test\node_modules\@nestjs\core\router\router-proxy.js:9:17
```

3. 비밀번호는 안전항 방식으로 저장되어야 합니다. 

- sha로 암호화 잘 되어있음