

## 09. 요구사항-2 로그인

### 1. 요구사항- 로그인

- 이메일과 비밀번호로 로그인할 수 있어야 합니다.
- 로그인 성공 시 JWT 토큰을 발급합니다.

### 2. 현재 매서드 정리

- `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;
}
```

- 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.');
```

```

}

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

```

- `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> {
    if (!password) {
      throw new Error('Password is required for hashing.');
```

```

    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);
  }
}

```

- `jwt.strategy.ts`

```

import { ExtractJwt, Strategy } from 'passport-jwt';
import { PassportStrategy } from '@nestjs/passport';
import { Injectable } from '@nestjs/common';

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor() {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: 'secretKey',
    });
  }

  async validate(payload: any) {
    return { userId: payload.sub, username: payload.username };
  }
}

```

- `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...
}

```

### 3. LoginDto 정의

LoginDto 는 이메일과 비밀번호 필드를 포함할 것입니다. 여기서 `class-validator` 라이브러리를 사용하여 필드에 대한 유효성 검사를 추가할 수 있습니다.

```

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

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

  @IsString()
  @IsNotEmpty()

```

```
password: string;
}
```

## 4. 로그인 로직 구현

AppController 의 login 메서드에서 LoginDto 를 사용하여 로그인 로직을 구현합니다.

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

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

  // ...

  @Post('login')
  async login(@Body() loginData: LoginDto) {
    return this.authService.login(loginData);
  }

  // ...
}
```

## 5. AuthService 에서 로그인 처리

AuthService 에서 login 메서드를 구현하여 로그인 로직을 완성합니다. 이메일과 비밀번호를 검증하고, 성공적으로 검증된 경우 JWT 토큰을 생성하여 반환합니다.

```
// auth.service.ts
@Injectable()
export class AuthService {
  constructor(
    private jwtService: JwtService,
    private userService: UserService // UserService 주입
  ) {}

  // ...

  async login(loginData: LoginDto) {
    const { email, password } = loginData;
    const user = await this.userService.findByEmail(email);
```

```

    if (!user) {
      throw new Error('사용자를 찾을 수 없습니다.');
```

```

    }

    const isPasswordValid = await this.comparePasswords(password, user.password);
    if (!isPasswordValid) {
      throw new Error('잘못된 비밀번호입니다.');
```

```

    }



    const payload = { username: user.email, sub: user.id };
    return {
      access_token: this.jwtService.sign(payload),
    };
  }

  // ...
}
```

위 코드에서 `login` 메서드는 사용자의 이메일과 비밀번호를 검증하고, 유효한 경우 JWT 토큰을 생성하여 반환합니다. `comparePasswords` 메서드는 제공된 비밀번호와 저장된 해시를 비교합니다.

이제 `login` 엔드포인트를 통해 사용자가 로그인할 수 있으며, 성공적인 로그인 시 JWT 토큰을 받을 수 있습니다.

## 6. 요구사항 체크

- 이메일과 비밀번호로 로그인할 수 있어야 합니다. 
- 로그인 성공 시 JWT 토큰을 발급합니다. 



http://localhost:3000/users/login

Save



POST

http://localhost:3000/users/login

Send

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

Cookies

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

Beautify

```
1 {  
2   "email": "example3@example.com",  
3   "password": "yourpassword"  
4 }  
5
```

Body Cookies Headers (7) Test Results



Status: 201 Created

Time: 114 ms

Size: 444 B



Save as example



Pretty

Raw

Preview

Visualize

JSON



```
1 {  
2   "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.  
   eyJ1c2VybmFtZSI6ImV4YW1wbGUzQGV4YW1wbGUuY29tIiwic3ViIjozLCJpYXQiOjE3MDYwODkwOTgsImV4cCI6MTcwNjA4OTE1OH0.  
   BgIXSQJ2NzOWxoDCm3Y1JZ9XITmHqPvCvIC1i1V3UyA"  
3 }
```