10. 요구사항-3 비밀번호 변경 API (실패)

1. 비밀번호 변경 API 요구사항 확인

- 로그인한 사용자는 비밀번호를 변경할 수 있어야 합니다.
- 새로운 비밀번호는 안전한 방식으로 저장되어야 합니다.

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';
import { LoginDto } from './dto/login.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) {
```

```
    return this.authService.login(loginData);
  }

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

- `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...
```

```
}
```

- `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 };
  }
}
```

- `auth.service.ts`

```
import { Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import * as bcrypt from 'bcrypt';
import { UserService } from './user.service';
import { LoginDto } from './dto/login.dto';

@Injectable()
export class AuthService {
  constructor(
    private jwtService: JwtService,
    private userService: UserService // UserService 주입
  ) {}

  // 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(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),
    };
  }

  // 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);
```

```
  }
}
```

3. 트러블 슈팅 - jwt 401 에러

- gpt가 짜준 코드 기반 authguard로 jwt 토큰을 인증하는 시스템을 구축했는데, 아무리 해도 jwt 인증이 401 에러가 나는 현상이 발생했다.
- 따라서 https://github.com/nestjs/nest 에 나와있는 공식 예제를 바탕으로 인증 시스템을 재구성하고자한다. 실제로 예제 다운받아서 설치해보면 정상적으로 jwt 인증 가능한 것을 볼 수 있다.

4. 회원가입 재구현

- 기존 `app.controller.ts` 에서 구현했던 것을 `/users/users.controller.ts` 으로 옮기도록 하자.