23. 리팩토링 - controller 로직을 service로

1. controller -> service

- 하다보니 controller에 로직이 복잡하게 들어있는 것들이 있다.
- 로직을 service로 빼자.

2. change-password 컨트롤러 리팩토링

- `@Patch('change-password')`

```
  @Patch('change-password')
  async changePassword(@Req() req, @Body() changePasswordDto: ChangePasswordDto) {
    const user = await this.usersService.findByUsername(req.user.username);
    if (!user) {
      throw new UnauthorizedException();
    }

    const { currentPassword, newPassword } = changePasswordDto;

    // 현재 비밀번호 확인
    const isPasswordValid = await
this.authService.comparePasswords(currentPassword, user.password);
    if (!isPasswordValid) {
      throw new Error('현재 비밀번호가 일치하지 않습니다.');
    }

    // 새 비밀번호 해싱 및 저장
    const hashedPassword = await this.authService.hashPassword(newPassword);
    await this.usersService.updatePassword(user.id, hashedPassword);

    return { message: '비밀번호가 변경되었습니다.' };
  }
```

2-1. 변경된 코드

```
@Patch('change-password')
async changePassword(@Req() req, @Body() changePasswordDto: ChangePasswordDto) {
        await this.authService.changePassword(req.user.username,
changePasswordDto.currentPassword, changePasswordDto.newPassword);
        return { message: '비밀번호가 변경되었습니다.' };
}
```

- 다음과 같이 깔끔하게 변경되었다.

3. getAllusers

`getAlluers(@Req req)`

```
  @Get()
  async getAllUsers(@Req() req) {
    // 사용자 역할 확인
    console.log(req.user.role)
    if (req.user.role !== UserRole.ADMIN) {
      throw new UnauthorizedException('관리자만 접근 가능합니다.');
    }

    // 모든 사용자 목록 조회
    const users = await this.usersService.findAll();
    return users;
  }
```

3-1. 변경된 코드

```
@Get()
async getAllUsers(@Req() req) {
        return await this.usersService.getAllUsers(req.user.role);
}
```

- 다음과 같이 아주 깔끔하게 바뀌었다.

4. register

- `register(@Body() registerDto: RegisterDto)`

```
  @Public()
  @Post('register')
  async register(@Body() registerDto: RegisterDto) {
    // 이메일 중복 검사
    const existingUser = await
this.usersService.findByUsername(registerDto.username);
    if (existingUser) {
      throw new Error('이미 존재하는 이메일입니다.');
    }

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

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

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

4. 변경된 코드

```
@Public()
@Post('register')
async register(@Body() registerDto: RegisterDto) {
        const user = await this.usersService.register(registerDto);
        return { username: user.username };
}
```

5. register를 service로 빼는 과정에서 생긴 오류

```
[Nest] 10296  - 2024. 01. 27. 오후 5:22:59   ERROR [ExceptionHandler] Nest can't
resolve dependencies of the AuthService (UserRepository, ?, JwtService). Please
```

```
make sure that the argument dependency at index [1] is available in the AuthModule
context.

Potential solutions:
- Is AuthModule a valid NestJS module?
- If dependency is a provider, is it part of the current AuthModule?
- If dependency is exported from a separate @Module, is that module imported within
AuthModule?
  @Module({
    imports: [ /* the Module containing dependency */ ]
  })

Error: Nest can't resolve dependencies of the AuthService (UserRepository, ?,
JwtService). Please make sure that the argument dependency at index [1] is
available in the AuthModule context.

Potential solutions:
- Is AuthModule a valid NestJS module?
- If dependency is a provider, is it part of the current AuthModule?
- If dependency is exported from a separate @Module, is that module imported within
AuthModule?
  @Module({
    imports: [ /* the Module containing dependency */ ]
  })

    at Injector.resolveSingleParam (C:\gits\ops-
test\node_modules\@nestjs\core\injector\injector.js:191:19)
    at resolveParam (C:\gits\ops-
test\node_modules\@nestjs\core\injector\injector.js:128:49)
    at Array.map (<anonymous>)
    at Injector.resolveConstructorParams (C:\gits\ops-
test\node_modules\@nestjs\core\injector\injector.js:143:58)
    at Injector.loadInstance (C:\gits\ops-
test\node_modules\@nestjs\core\injector\injector.js:70:24)
    at Injector.loadProvider (C:\gits\ops-
test\node_modules\@nestjs\core\injector\injector.js:97:20)
    at C:\gits\ops-test\node_modules\@nestjs\core\injector\instance-loader.js:56:33
    at Array.map (<anonymous>)
    at InstanceLoader.createInstancesOfProviders (C:\gits\ops-
test\node_modules\@nestjs\core\injector\instance-loader.js:55:36)
```

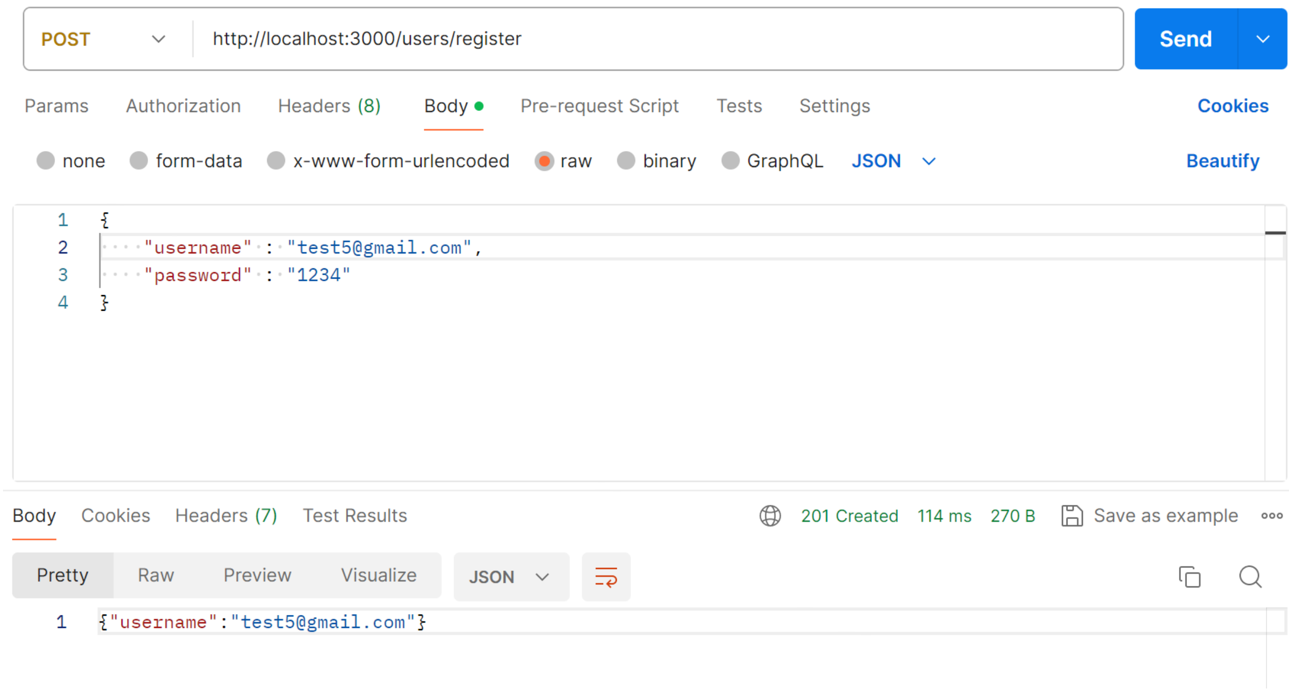
```
    at C:\gits\ops-test\node_modules\@nestjs\core\injector\instance-loader.js:40:24
PS C:\gits\ops-test>
```

- 엥? 갑자기 authService 에서 usersService를 불러올 수 없는 오류가 발생하였다.
- authService에서 usersService를 참조해야하고, usersService에선 또 authService를 참조해야하므로 모순이다. 이런 순환 오류를 해제하기 위해 다음과 같이 해결하자.
- forwardRef를 사용해서 순환 참조 오류를 해소할 수 있다.

```
// UsersService 내부
constructor(
  @InjectRepository(User)
  private readonly userRepository: Repository<User>,
  private emailService: EmailService,
  @Inject(forwardRef(() => AuthService))
  private authService: AuthService
) {}

// AuthService 내부
constructor(
  @InjectRepository(User)
  private readonly userRepository: Repository<User>,
  @Inject(forwardRef(() => UsersService))
  private usersService: UsersService,
  private jwtService: JwtService,
) {}
```

- 정상적으로 회원가입 기능 작동한다.



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

```
{
    "username" : "test5@gmail.com",
    "password" : "1234"
}
```

201 Created 114 ms 270 B

```
{"username":"test5@gmail.com"}
```

6. auth.controller

- 상당히 이뻐졌다.

```
@Controller('auth')
export class AuthController {
  constructor(private authService: AuthService,
              private usersService: UsersService) {}

  @Public()
  @HttpCode(HttpStatus.OK)
  @Post('login')
  signIn(@Body() signInDto: LoginDto) {
    return this.authService.signIn(signInDto.username, signInDto.password);
  }

  @Patch('change-password')
  async changePassword(@Req() req, @Body() changePasswordDto: ChangePasswordDto) {
    await this.authService.changePassword(req.user.username,
changePasswordDto.currentPassword, changePasswordDto.newPassword);
    return { message: '비밀번호가 변경되었습니다.' };
  }

  @Public()
  @Post('refresh')
  async refresh(@Body('refresh_token') refreshToken: string): Promise<{
access_token: string }> {
    return await this.authService.refreshAccessToken(refreshToken);
  }

  @Get('profile')
  getProfile(@Request() req) {
    return req.user;
  }
}
```

7. users.controller

- 상당히 이뻐졌다.

```
@Controller('users')
export class UsersController {
  constructor(private usersService: UsersService) { }

  @Get()
  async getAllUsers(@Req() req) {
    return await this.usersService.getAllUsers(req.user.role);
  }

  @Public()
  @Post('register')
  async register(@Body() registerDto: RegisterDto) {
    const user = await this.usersService.register(registerDto);
    return { username: user.username };
  }

  @Post('/sendcode')
  sendCode(@GetUser() userToken: User) {
    // 여기서 @GetUser는 jwt 토큰 자체 정보를 가져옴
    return this.usersService.sendVerificationCode(userToken);
  }

  @Post('/confirmcode')
  async confirmCode(@Body('verificationCode') code: string, @GetUser() userToken: User) {
    return { "isVerified": await this.usersService.confirmVerificationCode(code, userToken) };
  }

}
```