**데이트비 관리**

**프로그램 개발 일지**

****

• 2.19

기존 로그인 코드에서는 session에 “loginId”와 그 아이디에 해당하는 Member 변수를 기록하였다. 하지만 이런 코드는 한 명의 사용자만을 상대할 수 있고 여러 명의 사용자가 사용할려면 session의 키 값으로 고유한 값을 사용해야 했다.

그에 따라 생각한 **해결 방식은 session의 키 값으로 loginId를 직접 넣는 방법**이었다. 이런 방식을 사용할 경우에는 **클라이언트에서 자신의 Id에 대한 쿠키를 가지고 있어야만 로그인을 유지**하고 자신의 정보를 서버에서 언제든지 가져올 수 있었다.

• 2.22

에러를 리포지토리(DAO), 서비스 layer 등에서 중구난방식으로 발생시키면 나중에 유지보수를 할 때 너무 힘들어 질 수 있겠다는 생각을 하게 되었다(지금도 마찬가지이지만).

그래서 **모든 에러는 서비스 계층에서 발생**시키기로 하였다. 그리고 **세세한 에러 발생에 있어서는 각자의 컨트롤러에서 ExceptionHandler를 통해 해결**하기로 하였다. **공통 처리**가 필요한 에러는 아직까지는 발견하지는 못했지만 **ControllerAdvice를 통해서 해결**한다.

로그처리에 있어서 많은 고민을 해보았다. 모두 info로 찍기보다는 간단한 것들은 trace로, 사용자 입력에 대한 간단한 오류 처리는 info로 처리하기로 하였다.

(참고)

Error : 예상하지 못한 심각한 문제가 발생하는 경우, 즉시 조취를 취해야 할 수준의 레벨

Warn : 로직 상 유효성 확인, 예상 가능한 문제로 인한 예외 처리, 당장 서비스 운영에는 영향이 없지만 주의해야 할 부분

Info : 운영에 참고할만한 사항, 중요한 비즈니스 프로세스가 완료됨

Debug : 개발 단계에서 사용하며, SQL 로깅을 할 수 있음

Trace : 모든 레벨에 대한 로깅이 추적되므로 개발 단계에서 사용함

• 2.24

테스트 코드를 작성하였는데 이전에 개발했었던 MemoryMemberRepository에 대해서 수정을 안했던 것을 알았다. 오늘도 다시 한번 느끼지만 **처음에 코드 설계를 제대로 꼼꼼하고 세밀하게 해놓지 않는다면 나중에 개발하면서 수도 없이 코드를 수정하고 바꾸는 작업을 반복해야 한다**는 것을 느낀다.

테스트를 작성할 때 경험적인 느낌으로는 **@BeforeEach, AfterAll()과 같은 애노테이션이 붙은 메서드들은 제일 앞에 위치시켜서, 처음 테스트 코드를 분석하는 사람에게 인지시켜주어야** 할 것 같다.

• 2.27

처음으로 데이터베이스와 프로그램을 연동한 시간이었다. 아직 jdbcTemplate를 사용한 수준이었지만 오류가 어디서 무슨 익셉션이 발생할 지 예측할 수 없었다.   
가장 중요한 점은 처음에 설계를 제대로 하지 않았기 때문에 전면적으로 Service 계층을 수정해야 했다. 처음에 HashMap을 이용해서 memoryMemberRepository를 만들고 그에 적합한 service를 만들어서 인지, 중구난방식으로 프로그램이 만들어졌던 것 같다.

느낀점

1) 인터페이스 설계 시, 반환 값을 어떻게 줄지 또한 중요한 설계요소 포인트 인 것 같다.

2) **설계 시, 어떤 익셉션이 어떠한 형태로 어디서 발생될 것인지 미리 예측해야 한다. 그에 따라 서비스 계층과 컨트롤러 계층을 구성해야 한다.**

3) DAO 레벨도 테스트 코드가 필요하다.