

SW설계공학 프로젝트 결과보고서

본 보고서를 SW설계공학 교과목의 최종보고서로 제출합니다.

학과(전공): 컴퓨터소프트웨어공학

담당 교수: 이종민 교수

조 명: 2분반 6조

제 출 일: 2023년 05월 28일

조 장 : 김 민 곤 (20193172)

조원1 : 한 정 우 (20193175)

조원2 : 박 준 석 (20213070)

동 의 대 학 교

컴퓨터소프트웨어공학과장 귀하

〈 요약 문 〉

<p>프로젝트 목표 (300자내외)</p>	<p>이번 프로젝트의 주요 목표는 소프트웨어 설계 공학에서 배운 디자인 패턴을 적용하여 동의대학교 캠퍼스 맵 시스템을 구현하는 것이다. 디자인 패턴은 각각의 상황에 맞게 문제를 해결하기 위한 일련의 해결책들을 형성하는 것으로, 이 패턴들은 소프트웨어 설계에서 널리 사용되어 왔다. 이 프로젝트에서는 적절한 디자인 패턴을 적용하여 캠퍼스 맵 시스템의 기능을 최적화하고, 유지보수와 확장성이 용이한 소프트웨어를 만드는 것이 목표다. 이러한 목표를 달성하기 위해서는 팀원들 간의 협업과 역할 분담, 그리고 개발 과정에서 발생할 수 있는 문제를 빠르게 해결할 수 있는 능력이 필요하다. 최종적으로, 우리 팀은 효율적이고 성능 우수한 캠퍼스 맵 시스템을 개발하고, 이를 통해 소프트웨어 설계 공학에 대한 실제적인 경험을 쌓을 것이다.</p>		
<p>내용 (500자내외)</p>	<p>이번 캠퍼스 맵 시스템 프로젝트는 동의대학교 학생들과 교직원들이 쉽게 건물 정보와 위치를 찾아 이동할 수 있도록 돕는 시스템입니다. 이를 위해 다양한 기능을 제공합니다.</p> <p>먼저 로그인과 회원가입 기능을 제공하여 사용자 개인정보를 안전하게 관리할 수 있도록 한다. 로그인 후에는 지도 기능을 통해 캠퍼스 내 건물 정보를 확인할 수 있다. 지도 기능 외에도 건물 이름 검색 기능, 카테고리 기능 등 다양한 검색 기능을 제공하여 사용자들이 원하는 정보를 쉽게 찾을 수 있다.</p> <p>더 나아가서, 사용자들이 즐겨찾기로 등록한 건물만 보기 위한 즐겨찾기 검색 기능을 제공한다. 건물을 선택하면 해당 건물에 대한 자세한 안내를 제공하는 건물 안내 기능 또한 제공한다. 사용자들은 해당 건물에 대한 정보, 특징, 부대시설 등을 확인할 수 있다.</p> <p>또한 커뮤니티 서비스를 통해 여러 사용자 간에 정보를 공유할 수 있는 사용자 평가 기능도 제공합니다. 게시물 작성, 사용자 커뮤니티, 게시물 검색, 게시물 수정, 게시물 삭제 기능을 제공하여 자유롭게 정보를 공유하고 서로 소통할 수 있습니다.</p> <p>프로젝트에서는 모든 기능이 서로 상호작용 하도록 구현하여 사용자들에게 더욱 편리하고 유익한 서비스를 제공할 수 있도록 노력합니다. 이를 위해 소프트웨어 설계 공학에서 배운 디자인 패턴을 적용하고, 엄격한 코드 품질 관리, 빠른 실행력과 기능 구현 등을 실천합니다. 프로젝트 결과물은 사용자 요구 사항에 부합하는 만족스러운 결과를 추구하며, 이를 통해 사용자들을 위한 동의대 캠퍼스 생활을 더욱 스마트하고 편리하게 만들어 나갈 것입니다.</p>		
<p>기대효과 (200자내외)</p>	<p>이번 캠퍼스 맵 시스템 프로젝트는 사용자들이 캠퍼스 내 건물 정보를 쉽게 확인할 수 있도록 돕는데 그 목적이 있다. 이를 통해 사용자들이 효율적으로 시간을 활용할 수 있게 되어, 더욱 생산적인 동의대 캠퍼스 생활을 즐길 수 있다. 또한, 사용자 평가 기능을 제공하여 사용자들 간에 정보를 공유하고, 커뮤니티를 형성할 수 있다는 장점이 있다. 이를 통해 사용자들은 더욱 적극적으로 정보를 공유하고, 서로의 의견을 공유할 수 있게 된다. 프로젝트 결과물은 더욱 향상된 사용자 경험을 제공할 뿐만 아니라, 동의대학교 교직원, 학생, 방문객 모두를 대상으로 하는 유용한 정보 제공 서비스로 자리 잡을 것이다. 이를 통해 건강하고 활기찬 캠퍼스 문화가 발전할 것으로 기대가 된다.</p>		
<p>Keywords</p>	<p>캠퍼스 맵 시스템</p>	<p>지도</p>	<p>건물 안내</p>
	<p>사용자 평가</p>	<p>디자인 패턴</p>	<p>설계 공학</p>

목 차

1. 프로젝트 개요	3
1.1 비전	5
1.2 문제 기술	5
1.3 특징 목록	6
1.4 관련 기술	7
1.5 이해 당사자 요구사항	9
2. 시스템 분석	10
2.1 유스케이스 “회원가입”	12
2.2 유스케이스 “로그인”	14
2.3 유스케이스 “건물 검색”	16
2.4 유스케이스 “건물 안내”	18
2.5 유스케이스 “게시판”	20
2.5.1 유스케이스 “게시물 작성”	20
2.5.2 유스케이스 “게시물 조회·수정·삭제”	22
3. 시스템 설계 및 구현	24
3.1 옵저버 패턴(Observer Pattern) 적용	24
3.1.1 문제점	24
3.1.2 해결 방안	25
3.1.3 관련 코드 설명	26
3.1.3.1 Observer.java, Subject.java	26
3.1.3.2 BoardData.java	27
3.1.3.3 LoginData.java	28
3.1.3.4 MainForm.java	29
3.1.3.5 MainForm.java	30
3.2 전략 패턴(Strategy Pattern) 적용	31
3.2.1 문제점	31
3.2.2 해결 방안	31
3.2.3 관련 코드 설명	32
3.2.3.1 ConnectDB	32
3.2.3.2 ConnectMariaDB	32
3.2.3.3 ConnectMySQLdb	33
3.2.3.4 ConnectOracleDb	34
3.3 싱글턴 패턴(Singleton Pattern) 적용	35
3.3.1 문제점	35
3.3.2 해결 방안	35
3.3.3 관련 코드 설명	36
3.3.3.1 DataPool	36
3.3.3.2 ConnectionPool	36
3.4 데코레이터 패턴(Decorator Pattern) 적용	37
3.4.1 문제점	37
3.4.2 해결 방안	37

3.4.3 관련 코드 설명	38
3.4.3.1 DAO, DAOAbstract, BuildingDAO	38
3.4.3.2 DAODecorator	39
3.4.3.3 FindByNameBuildingDAO	40
3.5 빌더 패턴(Builder Pattern) 적용	41
3.5.1 문제점	41
3.5.2 해결 방안	41
3.5.3 관련 코드 설명	42
3.5.3.1 Board	42
3.5.3.2 BoardBuilder	43
4. 시스템 테스트	45
4.1 옵저버 패턴(Observer Pattern) 테스트	45
4.1.1 관련 테스트 코드	45
4.1.2 테스트 실행 결과	46
4.2 전략 패턴(Strategy Pattern) 테스트	47
4.2.1 관련 테스트 코드	47
4.2.2 테스트 실행 결과	47
4.3 싱글턴 패턴(Singleton Pattern) 테스트	48
4.3.1 관련 테스트 코드	48
4.3.2 테스트 실행 결과	48
4.4 데코레이터 패턴(Decorator Pattern) 테스트	49
4.4.1 관련 테스트 코드	49
4.4.2 테스트 실행 결과	49
4.5 빌더 패턴(Builder Pattern) 테스트	50
4.5.1 관련 테스트 코드	50
4.5.2 테스트 실행 결과	50
5. 프로젝트 결과	51
5.1 프로젝트 완성도	51
5.2 일정 계획 평가	52
5.2.1 WBS	52
5.2.2 CPM 네트워크	52
5.2.3 간트 차트	53
5.2.4 평가	53
5.3 형상 관리 평가	54
5.4 설계 구성요소	56
5.5 현실적 제한조건	57
6. 소감	58

1. 프로젝트 개요

1.1 비전

캠퍼스 맵 시스템 프로젝트에서 개발한 소프트웨어 시스템의 목적은 주로 사용자들이 캠퍼스 내 건물 정보를 쉽게 찾아 이동할 수 있도록 돕는 것이다. 이를 위해 지도 기능과 위치 안내, 건물 검색, 사용자 평가 등 다양한 기능을 제공하며, 이러한 기능들이 상호 작용하여 사용자에게 완전한 서비스를 제공할 수 있는 시스템을 목표로 한다.

이러한 소프트웨어 시스템의 가치는 다음과 같다. 첫째, 사용자들이 더욱 쉽고 효율적으로 건물 정보와 위치를 확인할 수 있다는 것이다. 이를 통해 불필요한 시간 낭비를 줄이고, 보다 더 생산적인 캠퍼스 내 생활을 만들어갈 수 있다. 둘째, 사용자 평가 기능과 커뮤니티 서비스를 제공하여 사용자들 간 정보를 자유롭게 공유할 수 있다는 것이다. 이를 통해 사용자들은 서로 피드백을 주고받아, 동의 대 캠퍼스 내에서 서로 도움을 줄 수 있는 지속적인 소통과 교류를 이룰 수 있습니다.

따라서, 캠퍼스 맵 시스템 프로젝트에서 개발한 소프트웨어는 사용자들의 캠퍼스 내 생활을 보다 효율적으로 만들어가는 역할을 할 것이다.

1.2 문제 기술

캠퍼스 맵 시스템 프로젝트에서 소프트웨어를 사용하여 해결하려는 주요 문제는 캠퍼스 내 건물과 시설의 정보와 위치를 빠르고 정확하게 찾아 이동하는데 걸리는 시간과 노력을 줄이는 것이다. 학생들, 교직원들, 그리고 방문객들이 쉽게 정보를 얻고, 효율적으로 이동할 수 있도록 돕기 위해 지도, 검색, 위치 안내 등 다양한 기능을 제공하는 소프트웨어를 개발하려고 한다.

또한, 사용자들 간의 정보 공유와 소통을 도모하기 위한 커뮤니티 기능을 통해, 캠퍼스 내 신속한 정보 전달과 더 나은 캠퍼스 경험을 제공할 수 있도록 하려고 한다. 이를 통해 소프트웨어를 사용하여 캠퍼스 생활의 효율성을 개선하고, 사용자들의 만족도를 높이는 것이 목표다.

1.3 특징 목록

연번	요구사항	추정치
SFR-1101	로그인 GUI를 띄우고 ID, PW를 입력받는다.	0.5
SFR-1102	로그인 버튼을 누르면 입력한 ID와 PW와 데이터베이스에 있는 정보가 일치하는지 확인 후 해당 정보로 로그인을 실행한다.	3
SFR-1201	회원가입 GUI를 띄우고 ID, PW, PW확인, 이름 등의 정보를 텍스트 필드로 입력받는다.	0.5
SFR-1202	ID 중복확인 버튼을 누르면 입력받은 ID가 기존에 존재하는 ID 인지 확인한다.	3
SFR-1203	회원가입 버튼을 누르면 필요한 모든 정보가 올바른지 입력되었는지 확인 후 입력한 정보를 데이터베이스에 저장한다.	3
SFR-2201	건물의 이름을 입력받는다.	0.5
SFR-2202	검색 버튼을 누르면 입력받은 이름에 맞는 건물을 선택한다.	1
SFR-3101	선택된 건물의 안내 화면을 띄운다.	3
SFR-3102	안내 화면 좌측에 건물 이미지를 출력하고 이미지의 아래쪽에 건물에 대한 정보를 출력한다.	8
SFR-3103	안내 화면 우측에 사용자들이 최근에 작성한 게시글들을 출력하는 리스트를 생성한다.	3
SFR-4101	게시글 작성 버튼을 누르면 게시글 작성 화면이 출력된다.	3
SFR-4102	제목, 내용을 텍스트 필드로 입력받고, 등록 버튼을 클릭하면 해당 게시물이 저장된다.	0.5
SFR-4103	저장한 데이터를 오피서버들에게 넘겨주어 최신 게시물 리스트에 업데이트 되도록 한다.	3
SFR-4201	안내 화면 또는 메인화면의 최근 게시물 리스트를 더블 클릭하면 해당 게시물을 조회한다.	3
SFR-4202	게시물의 조회수를 1 증가시키고 상태를 업데이트 한다.	2
SFR-4203	게시물에서 수정 및 삭제를 할 수 있으며, 수정 혹은 삭제 시도 시 변경된 정보를 저장하고 오피서버에 전달한다.	1
SFR-4301	사용자가 로그인 하였을 경우 게시글의 수정, 삭제, 작성이 가능하도록 한다.	3
SFR-4302	사용자의 상태 변화를 모든 오피서버에 전달한다.	1
SFR-4401	사용자가 로그인 한 상태일 때, 메인화면의 로그인 버튼이 로그아웃으로 바뀐다.	2
SFR-4402	로그아웃 버튼을 클릭하면 로그아웃이 되며 guest mode로 사용자의 상태가 변경된다.	2

1.4 관련 기술

연번	요구사항	관련 기술
SFR-1101	로그인 GUI를 띄우고 ID, PW를 입력받는다.	GUI (JAVA Swing)를 사용하여 표시해

		준다.
SFR-1102	로그인 버튼을 누르면 입력한 ID와 PW와 데이터베이스에 있는 정보가 일치하는지 확인 후 해당 정보로 로그인을 실행한다.	입력받은 데이터 정보를 MYSQL DB 서버 를 통해서 저장된 정보와 비교한다.
SFR-1201	회원가입 GUI를 띄우고 ID, PW, PW확인, 이름 등의 정보를 텍스트 필드로 입력받는다.	GUI (JAVA Swing) 를 사용하여 표시해 준다.
SFR-1202	ID 중복확인 버튼을 누르면 입력받은 ID가 기존에 존재하는 ID 인지 확인한다.	입력받은 데이터 정보를 MYSQL DB 서버 를 통해서 저장된 정보와 비교한다.
SFR-1203	회원가입 버튼을 누르면 필요한 모든 정보가 올바른지 입력되었는지 확인 후 입력한 정보를 데이터베이스에 저장한다.	회원 정보를 저장하기 위해 MYSQL DB 서버 를 사용한다.
SFR-2201	건물의 이름을 입력받는다.	N/A
SFR-2202	검색 버튼을 누르면 입력받은 이름에 맞는 건물을 선택한다.	DB에 저장된 데이터와 비교 후 반환한다.
SFR-3101	선택된 건물의 안내 화면을 띄운다.	N/A
SFR-3102	안내 화면 좌측에 건물 이미지를 출력하고 이미지의 아래쪽에 건물에 대한 정보를 출력한다.	N/A
SFR-3103	안내 화면 우측에 사용자들이 최근에 작성한 게시글들을 출력하는 리스트를 생성한다.	작성한 게시물의 조회하기 위해 MYSQL DB 서버 를 사용한다.
SFR-4101	게시글 작성 버튼을 누르면 게시글 작성 화면이 출력된다.	N/A
SFR-4102	제목, 내용을 텍스트 필드로 입력받고, 등록 버튼을 클릭하면 해당 게시물이 저장된다.	입력한 정보를 DB에 저장하고, 옵저버 패턴을 사용해 최신 게시글 리스트를 업데이트 한다.
SFR-4103	저장한 데이터를 옵저버들에게 넘겨주어 최신 게시글 리스트에 업데이트 되도록 한다.	입력한 정보를 DB에 저장하고, 옵저버 패턴을 사용해 최신 게시글 리스트를 업데이트 한다.
SFR-4201	안내 화면 또는 메인화면의 최근 게시글 리스트를 더블 클릭하면 해당 게시물을 조회한다.	N/A
SFR-4202	게시물의 조회수를 1 증가시키고 상태를 업데이트 한다.	옵저버 패턴 적용
SFR-4203	게시물에서 수정 및 삭제를 할 수 있으며, 수정 혹은 삭제 시도 시 변경된 정보를 저장하고 옵저버에 전달한다.	사용자 상태 변경을 모든 프레임에 전달하여 저장한다.
SFR-4301	사용자가 로그인 하였을 경우 게시글의 수정, 삭제, 작성이 가능하도록 한다.	"
SFR-4302	사용자의 상태 변화를 모든 옵저버에 전달한다.	"

SFR-4401	사용자가 로그인 한 상태일 때, 메인화면의 로그인 버튼이 로그아웃으로 바뀐다.	사용자의 상태 변화를 옵저버에 전달해준다.
SFR-4402	로그아웃 버튼을 클릭하면 로그아웃이 되며 guest mode로 사용자의 상태가 변경된다.	“

1.5 이해 당사자 요구사항

연번	이해 당사자	요구사항
St-01	시스템 기획자	기본적으로 지원되는 기능 외에 시스템 기획 시 추가적으로 요구되는 기능을 지원할 수 있어야 한다.
		모든 게시물에 대해 접근할 수 있으며, 추가, 수정, 삭제 권한을 가져야 한다.
		건물 정보가 변경되면 수정할 수 있어야 한다.
		건물을 선택하면 지도에 마커로 건물의 위치를 알 수 있어야 한다.
		검색을 통해 찾고자 하는 건물 정보를 검색할 수 있어야 한다.
		자주 가는 건물에 대한 정보를 등록, 삭제할 수 있어야 한다.
		선택한 건물의 정보를 조회할 수 있어야 한다.
St-02	사용자	사용자는 게시물을 조회, 등록할 수 있어야 한다.
		사용자는 본인이 작성한 게시물을 수정, 삭제할 수 있어야 한다.
		로그인, 회원가입 기능을 통해 사용자 정보를 수집, 저장할 수 있어야 한다.
		사용자가 로그인 할 시 DB 정보와 대조할 적절한 데이터를 입력받아야 한다.
		사용자가 데이터를 추가, 수정, 삭제할 경우, 서버 단에서 DB 서버에 데이터 접근을 요청한다.
		사용자 로그인시 사용자에게 맞는 화면을 보여줘야 한다.
		로그인했을 때와 게스트 모드일 때 기능이 달라야 한다.
St-03	시스템 개발자	권한에 맞는 화면을 볼 수 있어야 한다.
		회원가입 할 때 입력한 정보가 올바른지 확인 할 수 있어야 한다.
		사용자가 원하는 건물 정보를 사용자에게 안내하는 기능을 지원할 수 있어야 한다.
		검색 기능을 통해 사용자가 필요로 하는 정보를 제공할 수 있어야 한다.
		안내 화면에 사용자들이 최근에 작성한 게시글들을 출력하는 리스트를 생성할 수 있어야 한다.
		게시물 작성자가 게시물 수정 버튼을 클릭하면 게시물 제목과 내용이 변경 가능해야 한다.
		부적절한 데이터가 전송되지는 않는지 시스템이 지속적으로 확인해 주어야 한다.
		요청받은 서비스 내부적인 에러사항을 신속하게 처리할 수 있어야 한다.

2. 시스템 분석

유스 케이스	CampusMap System (프로그램 전체)	
액터	사용자, 시스템	
목적	사용자에게 캠퍼스 내 시설에 대한 위치 안내 및 정보를 제공한다.	
개요	사용자는 자신이 원하는 캠퍼스 건물에 대한 정보를 얻기 위해 프로그램을 사용한다. 프로그램은 사용자가 원하는 건물에 대한 위치 정보 및 건물에 대한 설명을 제공한다. 사용자는 검색 및 게시판 기능을 통해 보다 편리하게 프로그램을 이용할 수 있다.	
유형	핵심	
참조		
주 흐름 (main flow, basic flow)	액터	시스템
	<p>① 사용자가 캠퍼스 시설에 대한 정보를 얻기 위하여 프로그램을 실행시키면 유스 케이스가 시작된다.</p> <p>③ 사용자는 검색이나 건물 정보 테이블을 통해 원하는 건물을 찾을 수 있다.</p> <p>⑤ 사용자는 건물 정보 창에서 해당 건물의 게시글을 조회, 작성, 삭제, 수정할 수</p>	<p>② 시스템은 메인 화면을 표시한다. 메인 화면에는 전체 캠퍼스 맵 이미지, 최신 게시글, 건물 정보, 검색창, 로그인 버튼 등이 있다.</p> <p>④ 사용자가 선택한 건물의 정보 폼이 나타난다. 건물 정보 폼에는 게시글, 건물 정보, 건물 이미지 등이 있다.</p>

	있다.	⑥ 사용자가 게시글을 작성하고 등록 버튼을 누르면, 시스템은 해당 게시글의 제목 및 내용, 작성일, 유저 아이디 등을 저장한다. ⑦ 게시글을 수정, 삭제, 등록, 조회하면 게시글 목록에 상태를 업데이트한다.
대체 이벤트	⑤ 로그인 하지 않은 상태인 경우 로그인 하라는 메시지와 함께 로그인 폼이 나타난다.	

2.1 유스케이스 “회원가입”

유스 케이스	회원가입	
액터	사용자, 사용자 정보	
목적	사용자에게 데이터를 입력받아 정보를 userData에 저장한다.	
개요	사용자는 자신의 정보를 폼에 입력한다. 필수 정보들이 모두 입력되었는지 검사 후 회원가입을 승인한다. 회원가입에 성공하면 입력한 정보가 데이터베이스에 저장된다.	
유형	기능	
참조		
적용 패턴		
주 흐름 (main flow, basic flow)	액터	시스템
	<p>① 사용자가 회원가입 버튼을 누르면 유스케이스가 시작된다.</p> <p>③ 사용자는 id를 입력하고 중복확인 버튼을 클릭한다.</p>	<p>② 사용자에게 회원가입 폼을 보여준다.</p> <p>④ 데이터베이스에 중복되는 아이디가 있는지 검사한다.</p>

	<p>⑤ 사용 가능한 아이디라는 메시지를 보여준다.</p> <p>⑥ 사용자는 pw와 pw확인, 이름을 입력하고 등록 버튼을 누른다.</p> <p>⑦ pw와 pw확인란에 입력된 정보가 일치하는지 검사한다.</p> <p>⑧ 필요한 정보가 모두 입력되었는지 검사한다.</p> <p>⑨ 회원가입이 완료되고, 사용자가 입력한 정보가 데이터 베이스에 저장된다.</p>
대체 이벤트	<p>⑤ 중복되는 아이디가 있으면 이미 사용중이라는 메시지를 보여준다.</p> <p>⑦ pw와 pw확인이 일치하지 않으면 비밀번호가 일치하지 않는다는 메시지를 보여준다.</p> <p>⑧ 필수 입력 정보가 누락되어 있으면 정보를 입력해 달라는 메시지를 보여준다.</p>

2.2 유스케이스 “로그인”

유스 케이스	로그인	
액터	사용자, 사용자 정보	
목적	사용자에게 추가적인 기능을 이용할 수 있는 상태를 부여하기 위해 로그인 여부를 확인하고, 로그인 한 사용자의 상태를 시스템에 반영한다.	
개요	사용자는 자신의 정보를 폼에 입력한다. 시스템은 입력된 정보를 비교하고, 저장된 정보와 일치하면 로그인에 성공시키고, 바뀐 사용자 상태를 옵저버에 전송한다.	
유형	기능	
참조		
적용 패턴	<ul style="list-style-type: none"> - 옵저버 패턴을 사용하여, 사용자의 상태가 바뀌면 바뀐 상태를 저장하고 2.4와 2.5의 프레임 객체들에게 변경사항을 알려주어 업데이트한다. - 싱글톤 패턴을 사용하여 로그인 데이터가 하나의 객체만 생성하도록 하여 데이터 일관성을 유지하고 어느 위치에서든 접근 가능하도록 하였다. - 데코레이터 패턴을 사용하여 사용자의 정보를 추가, 수정, 삭제하는 작업을 수행할 때 기존 DAO를 받아와서 수행한다. 	
주 흐름 (main flow, basic flow)	액터	시스템
	① 사용자가 로그인 버튼을 누르면 유스케이스가 시작된다. ③ 사용자는 id와 pw를 입력하여 로그인을 요청한다.	② 사용자에게 로그인 폼을 보여준다.

	<p>④ id와 pw가 모두 입력되었는지 검사한다.</p> <p>⑤ 데이터베이스의 유저정보를 찾아와서 일치하면 로그인이 된다.</p> <p>⑥ 사용자 객체에 사용자 정보를 준다.</p> <p>⑦ 사용자 상태가 바뀌면 바뀐 상태를 저장하고 옵저버들에게 전송한다.</p> <p>⑧ 메인 폼의 로그인 버튼이 로그아웃으로 바뀌고, 건물 폼(게시판)의 사용자 상태가 바뀐다.</p>
대체 이벤트	<p>⑤ 일치하는 사용자가 없으면 id와 pw를 확인하라는 알림이 뜬다.</p>

2.3 유스케이스 “건물 검색”

유스 케이스	건물 검색	
액터	사용자, 게스트, 건물 정보	
목적	사용자가 원하는 건물을 찾을 수 있도록 한다.	
개요	사용자는 원하는 건물을 검색이나, 건물 정보 테이블을 통해 찾을 수 있다. 시스템은 사용자가 검색하거나 테이블에서 찾은 건물의 위치를 전체 캠퍼스 이미지에 마커로 표시해준다.	
유형	기능	
참조	<pre> graph LR 일반회원 --> 사용자 관리자 --> 사용자 사용자 --> 건물_안내((건물 안내)) guest --> 건물_안내 건물_정보[건물 정보] --> 검색((검색)) 검색 -.-> <<extends>> 건물_안내 </pre>	
적용 패턴	데코레이터 패턴을 구현한 추상 클래스를 상속받아 기본적으로 주어진 기능 외에 필요한 이름으로 데이터를 검색하는 메서드를 구현하였다.	
주 흐름 (main flow, basic flow)	액터	시스템
	<p>① 사용자가 프로그램을 실행하면 유스케이스가 시작된다.</p> <p>③ 사용자는 검색창을 이용하거나 건물 정보 테이블에서 원하는 건물을 찾는다.</p>	<p>② 사용자에게 메인화면 폼을 보여준다. 메인화면에는 최신 게시글, 건물 정보, 전체 캠퍼스 이미지, 검색 창, 로그인 버튼 등이 있다.</p>

	<p>⑥ 사용자가 건물 정보 테이블에서 더블 클릭을 하거나, 건물 정보 버튼을 누른다.</p>	<p>④ 건물 리스트에서 사용자가 선택한 인덱스에 해당하는 객체를 가져온다.</p> <p>⑤ 선택된 건물의 위치에 새로운 마커 라벨을 추가한다.</p> <p>⑦ 해당 건물 정보를 보여주는 건물 폼이 열린다.</p>
대체 이벤트	-	

2.4 유스케이스 “건물 안내”

유스케이스	건물 안내	
액터	사용자, 게스트, 건물/게시물 정보	
목적	사용자에게 건물에 대한 정보를 알려준다.	
개요	사용자에게 건물의 이미지와 설명, 해당 건물의 게시물을 보여준다. 데이터 변경 시 웹저버 인터페이스를 구현한 업데이트 메서드가 호출되어 게시물 목록을 업데이트한다.	
유형	기능	
참조	<pre> graph LR 일반회원 --> 사용자 관리자 --> 사용자 사용자 --> 건물_안내[건물 안내] guest --> 건물_안내 사용자_정보[사용자 정보] --> 건물_안내 건물_정보[건물 정보] --> 건물_안내 게시물_정보[게시물 정보] --> 게시물_목록[게시글 목록] 게시물_목록 -.-> <<extends>> 건물_안내 </pre>	
적용 패턴	웹저버 패턴을 사용하여 로그인 정보나 게시판 정보의 상태 변화가 생기면 이를 업데이트하여 UI를 변경한다.	
주 흐름 (main flow, basic flow)	액터	시스템
	① 사용자가 건물 정보 버튼을 클릭하면 유스케이스가 시작된다. ④ 데이터 변경 시 사용자 정보와 게시물	② 건물 정보와, 사용자 정보를 받아 해당 건물의 정보를 보여주는 창을 생성한다. ③ 현재 로그인한 사용자의 정보를 가져와 라벨에 표시한다.

	목록을 업데이트한다.	
대체 이벤트	③ 로그인 하지 않은 경우 "Guest Mode"로 표시된다.	

2.5 유스케이스 “게시판”

2.5.1 유스케이스 “게시물 작성”

유스케이스	게시물 작성	
액터	사용자, 게시글 정보	
목적	사용자가 건물에 대한 게시글을 작성하여 추가적인 정보나 리뷰를 남길 수 있게 한다.	
개요	사용자가 건물에 대한 게시글을 작성하고 등록할 수 있게 한다. 게시글에는 제목, 내용이 포함되어야 한다. 게시글이 등록되면 게시판의 상태를 업데이트한다.	
유형	기능	
참조	<pre> graph TD 일반회원 --> 사용자 관리자 --> 사용자 사용자 --> 게시판관리[게시판 관리] 사용자 --> 게시글목록조회[게시글 목록 조회] guest --> 게시글목록조회 삭제 --> 게시판관리 등록 --> 게시판관리 수정 --> 게시판관리 게시판관리 --> 게시글정보[게시글 정보] 게시글정보 --> 게시글목록조회 </pre>	
적용 패턴	<ul style="list-style-type: none"> - 옵저버 패턴을 사용하여 게시물을 작성하면 여러 프레임 객체들에게 변경 사항을 즉시 알려줘 업데이트한다. - 빌더 패턴을 사용하여 게시물 저장에 필요한 속성들을 모두 정의하고 속성의 값을 정해주는 기본 생성자와 빌더 객체를 받아온다. 	
주 흐름 (main flow, basic flow)	액터	시스템
	① 사용자가 게시글 작성 버튼을 클릭하면 유스케이스가 시작된다.	② 사용자에게 게시물을 작성할 수 있는 창을 보여준다. 폼에는 제목, 내용을 입력할 수 있는 공간과 등록, 취소 버튼이 있

	<p>③ 사용자가 게시글의 제목과 내용을 입력하고 버튼을 클릭한다.</p>	<p>다.</p> <p>④.1 사용자가 취소 버튼을 클릭하면 폼을 닫는다.</p> <p>④.2 사용자가 등록 버튼을 클릭하면 입력된 게시글 정보를 확인하고, 게시글을 등록한다.</p> <p>⑤ 게시물 목록의 상태를 업데이트하고 폼을 닫는다.</p>
대체 이벤트	<p>② 사용자가 로그인 하지 않은 상태라면 로그인을 하라는 메시지를 보여주고 로그인 폼이 나타난다.</p> <p>④.2 제목과 내용이 비어있는 경우 내용을 입력하라는 메시지가 나타난다.</p>	

2.5.2 유스케이스 “게시물 조회·수정·삭제”

유스케이스	게시물 조회·수정·삭제	
액터	사용자, 게시글 정보	
목적	사용자가 건물에 대한 게시글을 조회하고, 수정 및 삭제를 할 수 있도록 한다.	
개요	사용자가 건물에 대한 게시글의 조회, 수정, 삭제를 시도하면 게시판의 상태를 업데이트한다.	
유형	기능	
참조	<pre> graph TD subgraph Actors A1[일반회원] A2[관리자] A3[사용자] A4[guest] end subgraph UseCases UC1(삭제) UC2(등록) UC3(수정) UC4(게시판 관리) UC5(게시글 목록 조회) end subgraph Data D1(게시글 정보) end A1 --> A3 A2 --> A3 UC1 --> UC4 UC2 --> UC4 UC3 --> UC4 A3 --> UC4 A3 --> UC5 A4 --> UC5 UC4 --> D1 D1 --> UC5 </pre> <p>The diagram illustrates the actors (일반회원, 관리자, 사용자, guest) and their interactions with use cases (삭제, 등록, 수정, 게시판 관리, 게시글 목록 조회) and data (게시글 정보). Arrows show the flow of actions and data between these elements.</p>	
적용 패턴	<ul style="list-style-type: none"> - 옵저버 패턴을 사용하여 조회, 수정, 삭제를 시도할 경우 변경된 내용을 여러 프레임 객체에게 알려줘 즉시 업데이트한다. - 전략 패턴을 사용하여 데이터베이스의 정보를 변경, 추가, 삭제할 때 관련 코드를 쉽게 수정할 수 있도록 하여 확장성을 강화함. 	
주 흐름 (main flow, basic flow)	액터	시스템
	<p>① 사용자가 게시물을 클릭하면 유스케이스가 시작된다.</p>	<p>② 게시글의 조회수를 1 증가시키고, 게시글을 업데이트한다.</p> <p>③ 사용자에게 게시글의 제목, 내용, 작성</p>

	<p>자, 작성일 등을 표시하는 폼을 보여준다.</p> <p>④ 사용자는 게시물을 수정하거나 삭제를 시도한다.</p> <p>⑤.1 사용자가 삭제 버튼을 클릭한 경우 해당 게시글을 삭제하고 게시판의 상태를 업데이트하고 폼을 닫는다.</p> <p>⑤.2 사용자가 수정 버튼을 클릭한 경우 게시글 내용을 편집 가능한 상태로 설정한다. 닫기 버튼을 "저장"으로 변경한다.</p> <p>⑥ 사용자가 내용을 수정하고 저장 버튼을 눌러 수정을 시도한다.</p> <p>⑦ 수정된 게시글 내용을 저장한다.</p> <p>⑧ 사용자는 닫기 버튼을 클릭하여 폼을 닫는다.</p>
대체 이벤트	<p>④ 사용자가 로그인 하지 않은 상태라면 닫기 버튼만 나타나고, 수정 삭제를 시도할 수 없다.</p>

3. 시스템 설계 및 구현

3.1 옵저버 패턴(Observer Pattern) 적용

3.1.1 문제점

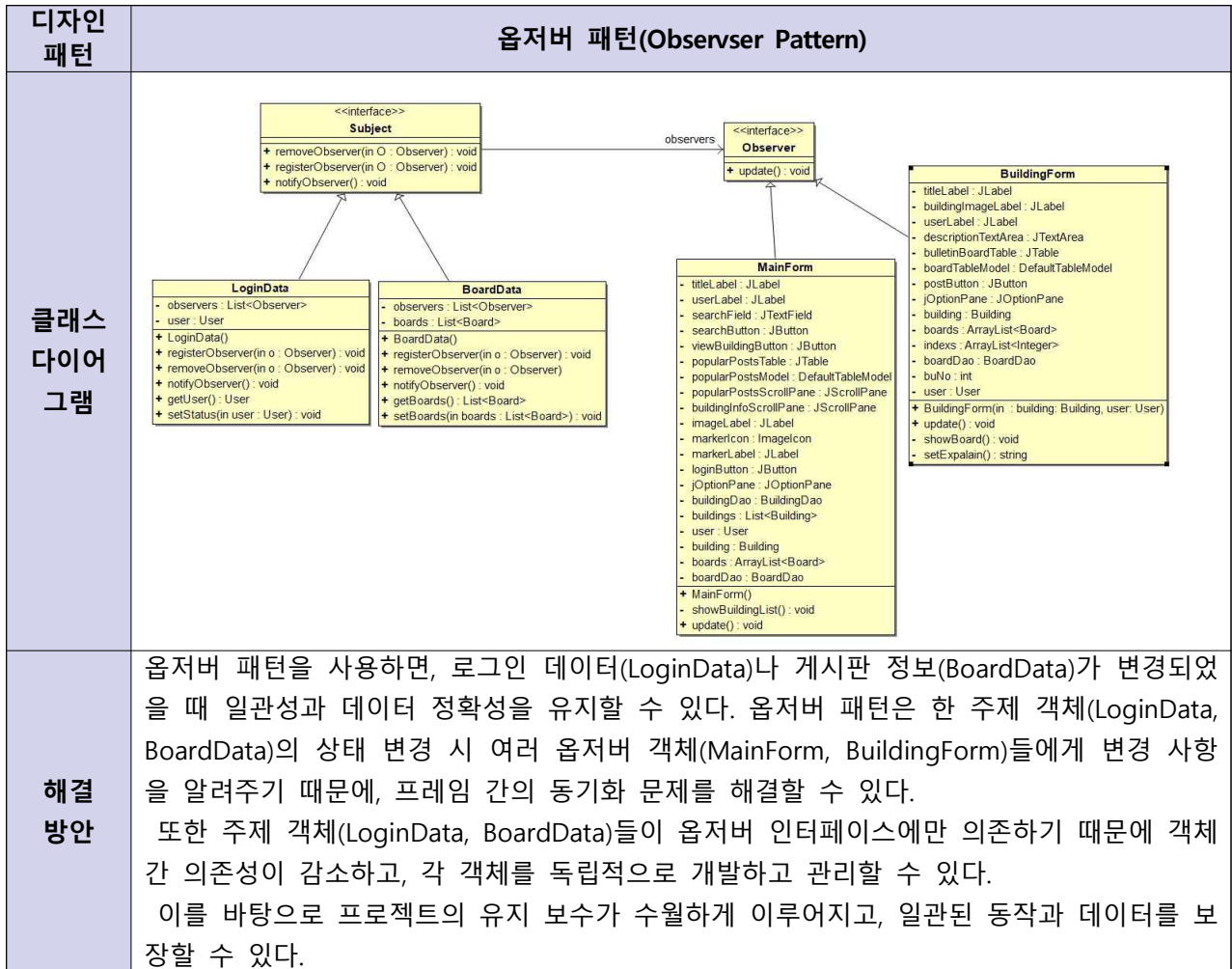
2.2절에서 기술한 로그인 기능과 2.5절에서 기술한 게시판 기능에서는 로그인 데이터가 변경되거나 게시판 정보가 변경되면 2.4절과 2.5절의 여러 프레임 및 객체들에게 변경 사항을 즉시 알려줘야 하는 일대다 의존 관계이다.

해당 기능을 구현할 때 옵저버 패턴을 사용하지 않은 구현에서는 다양한 프레임간의 동기화가 이루어지지 않아 일관성과 데이터 정확성의 문제가 예상되었다.

또한 여러 객체들과의 결합도가 높아져 코드의 수정 및 확장이 어려울 수 있는 문제점이 예상되었다. 데이터 변경 시 직접 다른 프레임과 객체들을 호출하는 방식은 유지 보수가 어려우며 코드 구조가 복잡해질 수 있다.

옵저버 패턴을 적용한 구현은 로그인 기능과 게시판 기능에서 발생할 수 있는 일관성, 데이터 정확성, 결합도와 관련하여 기술한 문제점들과 부합한다. 이를 통해 프레임간 동기화, 코드의 수정 및 확장성 문제, 유연한 유지 보수를 지원할 수 있는 구조를 가질 수 있다.

3.1.2 해결 방안



3.1.3 관련 코드 설명

3.1.3.1 Observer.java, Subject.java

Observer.java, Subject.java	
중요 코드	<pre>public interface Observer { public void update(); } public interface Subject { public void registerObserver(Observer o); public void removeObserver(Observer o); public void notifyObserver(); }</pre>
코드 설명	<p>Observer : 알림을 받을 옵저버 객체들이 공통적으로 가지고 있어야 하는 메서드인 update 메서드를 정의하여 후에 Observer 구현 클래스들이 update 메서드가 호출될 수 있게 한다.</p> <p>Subject: 주제 객체가 공통으로 가지고 사용할 메서드 registerObserver(), removeObserver(), notifyObserver()를 인터페이스로 정의하여 후에 Subject 구현 클래스들이 각각 옵저버 등록, 옵저버 제거, 옵저버 알림 기능으로 구현하여 사용할 수 있게 한다.</p>

3.1.3.2 BoardData.java

BoardData.java	
중요 코드	<pre> ... public class BoardData implements Subject { private List<Observer> observers; private List<Board> boards; public BoardData() { observers = new ArrayList<Observer>(); boards = new BoardDAO().findAll(); } @Override public void registerObserver(Observer o) { observers.add(o); } @Override public void removeObserver(Observer o) { observers.remove(o); } @Override public void notifyObserver() { for(Observer observer : observers){ observer.update(); } } public List<Board> getUser(){ return this.boards; } public void setStatus(List<Board> boards) { this.boards = boards; notifyObserver(); } } </pre>
코드 설명	<p>게시물 데이터 객체(Board)를 List 형태로 저장하고 있다가 setStatus(List<Board> boards) 메소드가 호출되면 boards 데이터를 저장하고 notifyObserver() 메서드를 호출하여 자신에게 등록된 모든 옵저버 객체들의 update() 메서드를 호출하여 게시물들의 데이터 변경 사실을 알린다.</p>

3.1.3.3 LoginData.java

LoginData.java	
중요 코드	<pre> ... public class LoginData implements Subject{ private List<Observer> observers; private User user; public LoginData() { observers = new ArrayList<Observer>(); user = null; } @Override public void registerObserver(Observer o) { observers.add(o); } @Override public void removeObserver(Observer o) { observers.remove(o); } @Override public void notifyObserver() { for(Observer observer : observers){ observer.update(); } } public User getUser(){ return this.user; } public void setStatus(User user) { this.user = user; notifyObserver(); } } </pre>
코드 설명	<p>현재 로그인 되어있는 사용자의 데이터 객체(User)를 저장하고 있다가 setStatus(User user) 메소드가 호출되면 user 데이터를 저장하고 notifyObserver() 메서드를 호출하여 자신에게 등록된 모든 옵저버 객체들의 update() 메서드를 호출하여 로그인 되어있는 사용자 데이터의 변경 사실을 알린다.</p>

3.1.3.4 MainForm.java

MainForm.java	
중요 코드	<pre> ... public class MainForm extends JFrame implements Observer { ... public MainForm() { ... DataPool.getInstance().getLoginData().registerObserver(this); DataPool.getInstance().getBoardData().registerObserver(this); ... } ... private void showBoard() { boards = (ArrayList) boardDao.findAll(); popularPostsModel.setNumRows(0); for (Board board : boards) { popularPostsModel.addRow(new Object[]{buildingDao .findById(board.getBdBuildNum()) .getBuName(), board.getUserName(), board.getBdTitle()}); } } ... @Override public void update() { this.user = DataPool.getInstance().getLoginData().getUser(); if (user != null) { loginButton.setText("로그아웃"); loginButton.setBackground(new Color(170, 105, 167)); userLabel.setText("User : " + user.getName()); } else { loginButton.setText("로그인"); loginButton.setBackground(new Color(125, 105, 167)); userLabel.setText("Guest Mode"); } showBoard(); } } </pre>
코드 설명	<p>registerObserver() 메서드를 사용해 생성자에서 자신 객체를 옵저버로 등록한다.</p> <p>update() 메서드가 호출될 경우 최신 데이터를 받아와 정보를 업데이트하고 객체에 적용시킨다.</p>

3.1.3.5 MainForm.java

BuildingForm.java	
중요 코드	<pre> ... public class MainForm extends JFrame implements Observer { ... public MainForm() { ... DataPool.getInstance().getLoginData().registerObserver(this); DataPool.getInstance().getBoardData().registerObserver(this); ... } ... private void showBoard() { boards = (ArrayList) boardDao.findAll(); popularPostsModel.setNumRows(0); for (Board board : boards) { popularPostsModel.addRow(new Object[]{buildingDao .findById(board.getBdBuildNum()) .getBuName(), board.getUserName(), board.getBdTitle()}); } } ... @Override public void update() { this.user = DataPool.getInstance().getLoginData().getUser(); if (user != null) { loginButton.setText("로그아웃"); loginButton.setBackground(new Color(170, 105, 167)); userLabel.setText("User : " + user.getName()); } else { loginButton.setText("로그인"); loginButton.setBackground(new Color(125, 105, 167)); userLabel.setText("Guest Mode"); } showBoard(); } } </pre>
코드 설명	<p>registerObserver() 메서드를 사용해 생성자에서 자신 객체를 옵저버로 등록한다.</p> <p>update() 메서드가 호출될 경우 최신 데이터를 받아와 정보를 업데이트하고 객체에 적용시킨다.</p>

3.2 전략 패턴(Strategy Pattern) 적용

3.2.1 문제점

데이터를 사용하는 모든 유스케이스에서 특정 데이터베이스에 강하게 결합된 구현이 만들어질 수 있다. 이러한 구현은 데이터베이스 변경, 추가 또는 제거 시 코드를 수정해야 하는 문제가 발생할 수 있어 코드의 유연성이 부족하며 각각의 클래스에서 데이터베이스 연결 코드가 중복될 수 있다. 이렇게되면 유사한 코드가 여러 곳에서 발견되고, 나중에 변경 시 각 클래스를 일일이 수정해야 하는 유지 보수의 어려움이 발생할 수 있다. 또한 데이터베이스의 종류가 늘어날 때마다 기존 코드를 일일이 수정하는 것은 비효율적이며, 새로운 데이터베이스 연결 방식을 쉽게 구현할 수 없게 된다.

전략 패턴은 접근 방식이 다른 여러 알고리즘(여기서는 다양한 데이터베이스 연결 방식)을 선택할 수 있도록 한다. 따라서 이 패턴은 위에서 언급한 문제점들에 유효한 해결 방안을 제공할 수 있다.

3.2.2 해결 방안

디자인 패턴	전략 패턴(Strategy Pattern)
클래스 다이어그램	<pre> classDiagram class ConnectDB { <<interface>> +getConn() Connection } class ConnectMariaDB { -JDBC_DRIVER String -DB_URL String -USER String -conn Connection +ConnectMariaDB() } class ConnectMySQLDb { -JDBC_DRIVER String -jdbcDriver String -dbUser String -dbPass String -conn Connection +ConnectMySQLDb() } class ConnectOracleDb { -JDBC_DRIVER String -DB_URL String -USER String -PW String -conn Connection +ConnectOracleDb() } ConnectDB < .. ConnectMariaDB ConnectDB < .. ConnectMySQLDb ConnectDB < .. ConnectOracleDb </pre>
해결 방안	<p>전략 패턴을 사용하여 각 데이터베이스 연결 방식을 독립적인 전략으로 구현하므로, 데이터베이스 변경, 추가 또는 제거 시 관련 코드를 손쉽게 수정할 수 있게 된다.</p> <p>이렇게 함으로써 프로젝트의 유연성이 향상될 수 있고, 데이터베이스 연결 코드를 하나의 전략으로 분리시켜 각 클래스에서 중복 코드를 제거하고, 변경 시 일일이 코드를 수정할 필요가 없어진다.</p> <p>또한 전략 패턴을 사용하면, 새로운 데이터베이스 연결 방식을 쉽게 구현할 수 있다. 기존 코드를 수정하지 않고 인터페이스 구현만으로 새로운 전략을 추가할 수 있기 때문에 프로젝트의 확장성이 강화된다.</p> <p>따라서 전략 패턴은 데이터베이스 연결에 관한 유연성 부족, 중복 코드, 확장성 제한 등의 문제점에 대한 적절한 해결 방안을 제공하여 프로젝트를 개선할 수 있다.</p>

3.2.3 관련 코드 설명

3.2.3.1 ConnectDB

ConnectDB.java	
중요 코드	<pre>public interface ConnectDB { public Connection getConn(); }</pre>
코드 설명	Connection 객체를 반환하는 getConn메서드를 인터페이스로 정의하여 해당 인터페이스를 구현한 클래스들이 공통적으로 getConn 메서드를 사용할 수 있게 한다.

3.2.3.2 ConnectMariaDB

ConnectMariaDB.java	
중요 코드	<pre>public class ConnectMariaDB implements ConnectDB { private static final String JDBC_DRIVER = "org.mariadb.jdbc.Driver"; private static final String DB_URL = "jdbc:mariadb://113.198.235.241:9090/campusapdb?useUnicode=true&characterEncoding=utf8"; private static final String USER = "gon"; private static final String PW = "20193172"; private static Connection conn; public ConnectMariaDB() { //DB 연결 if (conn == null) { try { Class.forName(JDBC_DRIVER); conn = DriverManager.getConnection(DB_URL, USER, PW); } catch (ClassNotFoundException SQLException e) { throw new RuntimeException(e); } } } @Override public Connection getConn() { return conn; } }</pre>
코드 설명	생성자에서 MariaDB에 연결하여 connection 객체를 받아 getConn() 메서드가 호출될 때 conn객체를 반환한다.

3.2.3.3 ConnectMySQLDb

ConnectMySQLDb.java	
중요 코드	<pre> public class ConnectMySQLDb implements ConnectDB{ private static final String JDBC_DRIVER = "com.mysql.jdbc.Driver"; String jdbcDriver = "jdbc:mysql://49.50.174.5:3306/TestDB?serverTimezone=UTC"; String dbUser = "gon"; //mysql id String dbPass = "20193172"; //mysql password private static Connection conn; public ConnectMySQLDb() { if (conn == null) { try { Class.forName(JDBC_DRIVER); conn = DriverManager.getConnection(jdbcDriver, dbUser, dbPass); } catch (ClassNotFoundException SQLException e) { throw new RuntimeException(e); } } } @Override public Connection getConn() { return conn; } } </pre>
코드 설명	<p>생성자에서 MySQLDB에 연결하여 connection 객체를 받아 getConn() 메서드가 호출될 때 conn객체를 반환한다.</p>

3.2.3.4 ConnectOrcleDb

ConnectOrcleDb.java	
중요 코드	<pre> public class ConnectOrcleDb implements ConnectDB { //DB 관리 private static final String JDBC_DRIVER = "oracle.jdbc.driver.OracleDriver"; private static final String DB_URL = "jdbc:oracle:thin:@sedb.deu.ac.kr:1521/orcl"; private static final String USER = "a20193172"; private static final String PW = "20193172"; private static Connection conn; public ConnectOrcleDb() { //DB 연결 if (conn == null) { try { Class.forName(JDBC_DRIVER); conn = DriverManager.getConnection(DB_URL, USER, PW); } catch (ClassNotFoundException SQLException e) { throw new RuntimeException(e); } } } @Override public Connection getConn() { return conn; } } </pre>
코드 설명	<p>생성자에서 OracleDB에 연결하여 connection 객체를 받아 getConn() 메서드가 호출될 때 conn객체를 반환한다.</p>

3.3 싱글턴 패턴(Singleton Pattern) 적용

3.3.1 문제점

데이터를 사용하는 모든 유스케이스에서 데이터베이스 커넥션 객체와 옵저버 패턴의 주제 객체인 현재 사용자 데이터 객체, 게시물 데이터 객체 등이 중복 생성되거나 제한된 객체 접근에 문제가 생길 수 있다.

여러 군데에서 객체를 생성할 때, 그 객체들이 중복되어 메모리 낭비와 데이터 관리에 어려움을 초래할 수 있고 이로 인해 프로그램 성능이 저하되거나 데이터 일관성이 손상될 위험이 있다.

또한 전역 접근이 필요한 객체를 각각 사용할 때, 전역 객체가 아닌 경우 객체 접근이 어려울 수 있다. 이로 인해 여러 컴포넌트에서 동일한 객체를 공유하고 사용하기 어렵게 된다.

싱글턴 패턴은 이러한 문제점을 해결할 수 있다.

3.3.2 해결 방안

디자인 패턴	싱글턴 패턴(Singleton Pattern)
클래스 다이어그램	<div style="border: 1px solid black; padding: 5px; margin-bottom: 20px;"> <p style="text-align: center;">ConnectionPool</p> <ul style="list-style-type: none"> - <u>instance : ConnectionPool</u> - <u>connection : ConnectionPool</u> + <u>getInstance(in connectDB : ConnectDB) : ConnectionPool</u> + <u>getConnection() : Connection</u> </div> <div style="border: 1px solid black; padding: 5px;"> <p style="text-align: center;">DataPool</p> <ul style="list-style-type: none"> - <u>instance : DataPool</u> - <u>loginData : LoginData</u> - <u>boardData : BoardData</u> - <u>DataPool()</u> + <u>getInstance() : DataPool</u> + <u>getBoardData() : BoardData</u> + <u>getLoginData() : LoginData</u> </div>
해결 방안	<p>싱글턴 패턴을 사용하면, 프로그램 전체에서 단 하나의 객체만 생성되도록 보장한다. 객체를 호출하는 곳에서도 항상 동일한 객체가 반환되므로 메모리 낭비를 줄이고 데이터 일관성을 유지할 수 있다.</p> <p>싱글턴 패턴을 통해 생성된 인스턴스는 전역에서 공유되기 때문에, 프로젝트의 어느 위치에서든 접근이 가능해진다. 이로 인해 여러 컴포넌트가 효과적으로 동일한 객체에 접근할 수 있게 된다.</p> <p>따라서, 싱글턴 패턴을 사용하여 프로젝트에서 예상되었던 문제점들을 해결할 수 있으며, 이 패턴의 적용이 문제점에 부합합니다. 이를 통해 객체 생성과 접근에 대한 유지 보수가 용이하게 되고, 프로그램 성능과 데이터 일관성이 개선된다.</p>

3.3.3 관련 코드 설명

3.3.3.1 DataPool

DataPool.java	
중요 코드	<pre>public class DataPool { private static DataPool instance; private LoginData loginData; private BoardData boardData; private DataPool() { loginData = new LoginData(); boardData = new BoardData(); } public static DataPool getInstance() { if (instance == null) { instance = new DataPool(); } return instance; } public BoardData getBoardData() { return boardData; } public LoginData getLoginData() { return loginData; } }</pre>
코드 설명	DataPool 객체가 하나만 존재하도록 하기 위해 자기 자신을 private static 멤버로 선언하고 생성자의 접근제어자를 private로 설정해 외부에서 생성하지 못하도록 한다. 이후 외부에서 멤버로 선언된 instance를 가져올 수 있는 메서드인 getInstance를 만들어 instance를 반환 받을 수 있게 하였다.

3.3.3.2 ConnectionPool

ConnectionPool.java	
중요 코드	<pre>public class ConnectionPool { private static ConnectionPool instance; private Connection connection; private ConnectionPool(ConnectDB connectDB) { this.connection = connectDB.getConn(); } public static ConnectionPool getInstance(ConnectDB connectDB) { if (instance == null) { instance = new ConnectionPool(connectDB); } return instance; } public Connection getConnection() { return connection; } }</pre>
코드 설명	ConnectionPool 객체가 하나만 존재하도록 하기 위해 자기 자신을 private static 멤버로 선언하고 생성자의 접근제어자를 private로 설정해 외부에서 생성하지 못하도록 한다. 이후 외부에서 멤버로 선언된 instance를 가져올 수 있는 메서드인 getInstance를 만들어 instance를 반환받을 수 있게 하였다.

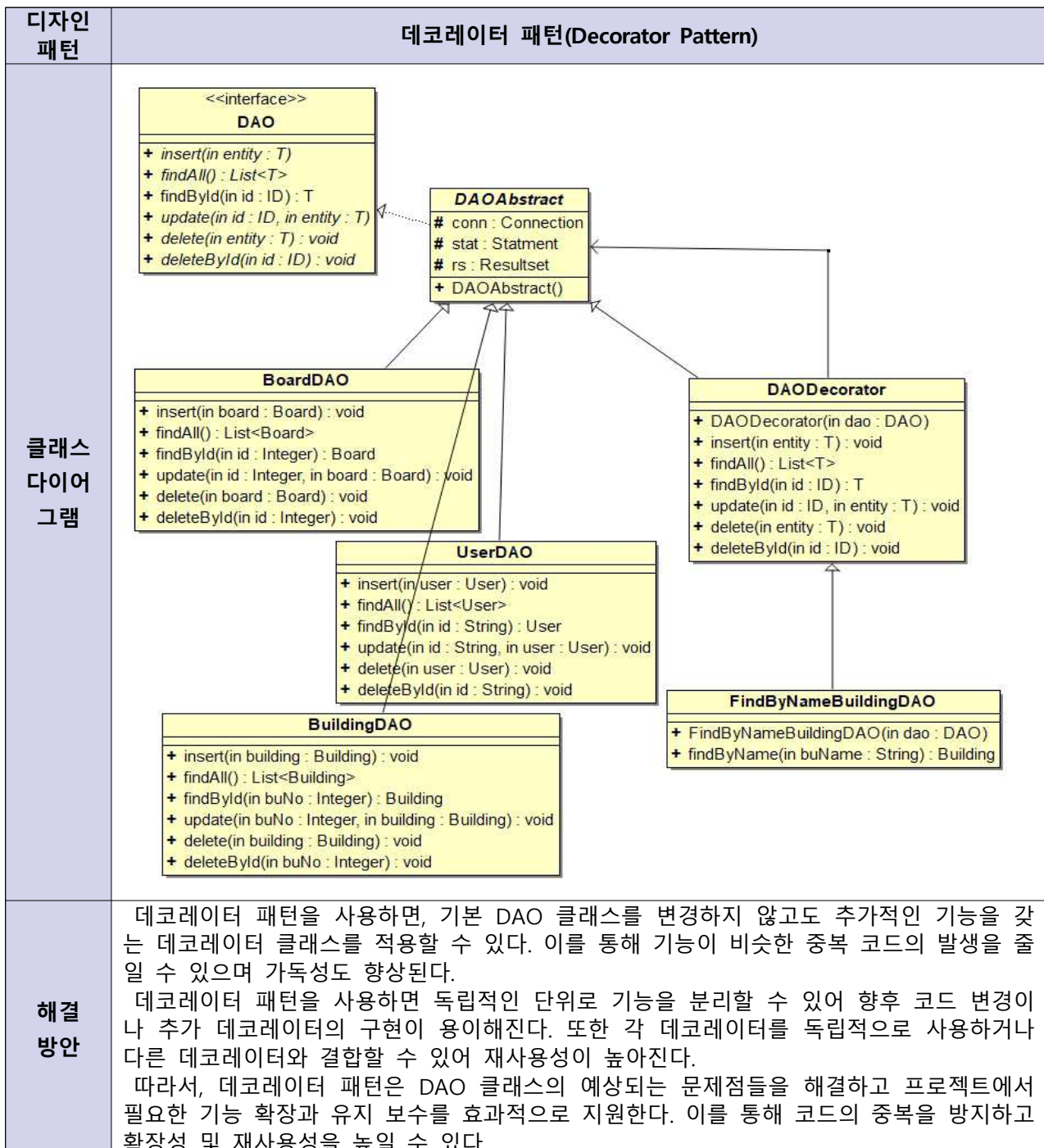
3.4 데코레이터 패턴(Decorator Pattern) 적용

3.4.1 문제점

2.3절에서 기술한 유스케이스와 향후에 추가될 기능에서 기본적인 데이터 처리 로직 외에 추가적인 기능이 필요한 경우, 각 기능별로 새로운 DAO 구현 클래스를 작성해야 한다. 이로 인해 유사한 코드가 여러 곳에서 반복되고 중복될 수 있다.

기존의 클래스를 변경해야하는 경우, 변경에 따른 오류 발생 가능성이 커지고 시간과 노력이 소요된다. 또한 기능 추가나 변경이 필요할 때마다 이를 포함하는 클래스를 생성할 때 중복되거나 재사용이 어려운 코드가 발생할 수 있다.

3.4.2 해결 방안



3.4.3 관련 코드 설명

3.4.3.1 DAO, DAOAbstract, BuildingDAO

DAO.java, DAOAbstract.java, BuildingDAO.java	
중요 코드	<pre>public interface DAO<T, ID>{ public abstract void insert(T entity); public abstract List<T> findAll(); public abstract T findById(ID id); public abstract void update(ID id, T entity); public abstract void delete(T entity); public abstract void deleteById(ID id); } public abstract class DAOAbstract<T, ID> implements DAO<T, ID> { protected Connection conn; protected Statement stat = null; protected ResultSet rs = null; public DAOAbstract() { conn = ConnectionPool.getInstance(new ConnectMariaDB()) .getConnection(); try { stat = conn.createStatement(); } catch (SQLException ex) { } } } public class BuildingDAO extends DAOAbstract<Building, Integer> { ... }</pre>
코드 설명	<p>DAO 인터페이스에서 기본적인 데이터 처리를 담당하는 메서드들을 정의해 해당 인터페이스를 구현하는 클래스들이 공통적으로 해당 기능을 사용하게 하였다.</p> <p>DAOAbstract 추상 클래스에서 데이터 처리 로직에서 필요한 속성들을 정의하고 해당 속성의 값을 설정하는 생성자를 정의하였다.</p> <p>BuildingDAO 구상 클래스에서 DAOAbstract를 상속받아 기본적인 데이터 처리 로직들을 모두 구현하였다.</p>

3.4.3.2 DAODecorator

DAODecorator.java	
중요 코드	<pre> public abstract class DAODecorator<T, ID> extends DAOAbstract<T, ID> { DAO<T, ID> dao; public DAODecorator(DAO dao) { this.dao = dao; } @Override public void insert(T entity) { dao.insert(entity); } @Override public List<T> findAll() { return dao.findAll(); } @Override public T findById(ID id) { return dao.findById(id); } @Override public void update(ID id, T entity) { dao.update(id, entity); } @Override public void delete(T entity) { dao.delete(entity); } @Override public void deleteById(ID id) { dao.deleteById(id); } } </pre>
코드 설명	<p>DAOAbstract를 상속받아 구현한 컴포넌트의 기본적인 데이터 처리 로직들을 데코레이터 클래스에서 사용할 수 있게 컴포넌트의 인스턴스(dao)를 받아와 자신의 속성에 저장하고 dao를 이용하여 기본적인 데이터 처리 로직을 구현하였다.</p>

3.4.3.3 FindByNameBuildingDAO

FindByNameBuildingDAO.java	
중요 코드	<pre> public class FindByNameBuildingDAO extends DAODecorator<Building, Integer> { public FindByNameBuildingDAO(DAO dao) { super(dao); } public Building findByName(String buName) { Building building = null; ArrayList<BuInfo> buInfos = new ArrayList(); try { String format = "SELECT * FROM %s WHERE buName = '%s'"; String query = String.format(format, "BUILDING", buName); System.out.println(query); rs = stat.executeQuery(query); while (rs.next()) { building = new Building(rs.getInt("buNo"), rs.getString("buName"), rs.getString("buExplain"), rs.getInt("buLocateX"), rs.getInt("buLocateY"), rs.getString("buImage"), rs.getInt("buFavo")); } if (building != null) { String biFormat = "SELECT * FROM %s WHERE buNo = %d"; query = String.format(biFormat, "BUINFO", building.getBuNo()); Statement biStat = conn.createStatement(); ResultSet rsTemp = biStat.executeQuery(query); while (rsTemp.next()) { BuInfo buInfo = new BuInfo(rsTemp.getInt("buNo"), rsTemp.getString("biFloor"), rsTemp.getString("biName")); buInfos.add(buInfo); } building.setBuInfos(buInfos); } } catch (SQLException ex) { } return building; } } </pre>
코드 설명	<p>DAODecorator를 상속받아 구현한 데코레이터 클래스이다. 기본적으로 주어진 기능 외에 추가적으로 필요한 이름으로 데이터를 검색하는 findByName() 메서드를 구현하였다.</p>

3.5 빌더 패턴(Builder Pattern) 적용

3.5.1 문제점

2.5절에서 기술한 게시판 기능에서 게시물의 정보를 저장하는 객체의 속성이 많을 경우, 각 속성에 대한 생성자를 작성하고 관리하는 것이 복잡해진다. 이렇게 되면 가독성이 떨어지고 코드 유지 보수가 어려워 질 수 있고 모든 속성을 생성자로 받아오지 않고, 일부 속성만 초기화할 수 있는 방법이 없습니다. 이로 인해 일부 속성만 초기화 해야하는 경우에도 모든 속성을 초기화해야 하는 제약이 발생할 수 있다.

기본 생성자와 매개변수 생성자를 사용한 경우, 코멘트가 없다면 각 매개변수의 의미를 파악하기 어려울 수 있다. 이로 인해 코드의 가독성이 떨어지게 된다.

빌더 패턴을 사용하면 이러한 문제점들을 해결할 수 있다.

3.5.2 해결 방안

디자인 패턴	빌더 패턴(Builder Pattern)
클래스 다이어그램	<pre> classDiagram class Board { -userId : String -userName : String -bdTitle : String -bdContent : String -bdDate : Timestamp -bdViewCnt : int -bdBuildNum : int -bdNo : int +Board(String, String, String, String, Timestamp, int, int, int) +Board(BoardBuilder) +getBdBuildNum() : int +setBdBuildNum(int) : void +getUserId() : String +setUserId(String) : void +getUserName() : String +setUserName(String) : void +getBdTitle() : String +setBdTitle(String) : void +getBdContent() : String +setBdContent(String) : void +getBdDate() : Timestamp +setBdContent(String) : void +getBdViewCnt() : int +setBdViewCnt(int) : void +getBdNO() : int +setBdNo(int) : void +toString() : String } class BoardBuilder { -userId : String -userName : String -bdTitle : String -bdContent : String -bdDate : Timestamp -bdViewCnt : int -bdBuildNum : int -bdNo : int +BoardBuilder(String, String, String, String) +bdDate(Timestamp) : BoardBuilder +bdViewCnt(int) : BoardBuilder +bdBuildNum(int) : BoardBuilder +bdNo(int) : BoardBuilder +build() : Board +getUserId() : String +getUserName() : String +getBdTitle() : String +getBdContent() : String +getBdDate() : Timestamp +getBdViewCnt() : int +getBdBuildNum() : int +getBdNo() : int } BoardBuilder --> Board </pre>
해결 방안	<p>빌더 패턴을 사용하면, 객체를 조립하기 위한 중첩된 빌더 클래스를 생성하여 복잡한 생성자를 간결하게 만들 수 있다. 이로 인해 코드 가독성과 유지 보수가 향상되고 개별 속성을 초기화할 수 있는 함수를 제공하여 필요한 속성만 선택적으로 초기화할 수 있다. 이렇게 함으로써 객체 생성에 대한 유연성이 향상된다.</p> <p>또한 빌더 패턴을 사용하면 setter 메소드를 통해 속성 이름을 명확하게 표시할 수 있으므로 코드의 가독성이 향상된다. 이로 인해 각 속성의 의미를 쉽게 파악할 수 있으며 유지 보수가 용이해진다.</p> <p>따라서, 빌더 패턴을 사용하여 게시물 정보 객체의 예상되었던 문제점들을 해결할 수 있으며, 이 패턴의 적용이 문제점에 부합하다고 할 수 있다. 이를 통해 객체 생성에 대한 유연성, 가독성, 유지 보수성이 개선된다.</p>

3.5.3 관련 코드 설명

3.5.3.1 Board

Board.java	
중요 코드	<pre>public class Board { private String userId; private String userName; private String bdTitle; private String bdContent; private Timestamp bdDate; private int bdViewCnt; private int bdBuildNum; private int bdNo; public Board(String userId, String userName, String bdTitle, String bdContent, Timestamp bdDate, int bdViewCnt, int bdBuildNum, int bdNo) { this.userId = userId; this.userName = userName; this.bdTitle = bdTitle; this.bdContent = bdContent; this.bdDate = bdDate; this.bdViewCnt = bdViewCnt; this.bdBuildNum = bdBuildNum; this.bdNo = bdNo; } public Board(BoardBuilder builder) { this.userId = builder.getUserId(); this.userName = builder.getUserName(); this.bdTitle = builder.getBdTitle(); this.bdContent = builder.getBdContent(); this.bdDate = builder.getBdDate(); this.bdViewCnt = builder.getBdViewCnt(); this.bdBuildNum = builder.getBdBuildNum(); this.bdNo = builder.getBdNo(); } ... }</pre>
코드 설명	게시물 정보 저장에 필요한 속성들을 모두 정의하고 해당 속성의 값을 정해주는 기본 생성자와 BoardBuilder 객체를 받아와 해당 빌더에 저장되어있는 값을 받아오는 생성자를 정의하였다.

3.5.3.1 BoardBuilder

BoardBuilder.java	
중요 코드	<pre>public class BoardBuilder { private String userId; private String userName; private String bdTitle; private String bdContent; // Set optional attributes private Timestamp bdDate; private int bdViewCnt; private int bdBuildNum; private int bdNo; public BoardBuilder(String userId, String userName, String bdTitle, String bdContent) { this.userId = userId; this.userName = userName; this.bdTitle = bdTitle; this.bdContent = bdContent; bdViewCnt = 0; } public BoardBuilder bdDate(Timestamp bdDate) { this.bdDate = bdDate; return this; } public BoardBuilder bdViewCnt(int bdViewCnt) { this.bdViewCnt = bdViewCnt; return this; } public BoardBuilder bdBuildNum(int bdBuildNum) { this.bdBuildNum = bdBuildNum; return this; } public BoardBuilder bdNo(int bdNo) { this.bdNo = bdNo; return this; } public Board build() { return new Board(this); } public String getUserId() { return userId; } public String getUserName() { return userName; } public String getBdTitle() { return bdTitle; } public String getBdContent() { return bdContent; } public Timestamp getBdDate() { return bdDate; } public int getBdViewCnt() {</pre>

	<pre> return bdViewCnt; } public int getBdBuildNum() { return bdBuildNum; } public int getBdNo() { return bdNo; } } </pre>
코드 설명	<p>게시물 정보 저장에 필요한 속성들을 모두 정의하고 해당 속성들을 필수적으로 필요한 속성과 필수적이지 않은 속성으로 나누었다.</p> <p>꼭 필요한 속성들은 생성자에서 인자로 받아 저장하고 필수적이지 않은 속성들은 속성 이름의 메소드를 사용해 값을 받아온다.</p> <p>원하는 값을 모두 넣었을 경우 Build 메서드를 호출하여 자신을 Board 생성자의 인자로 전달하여 Board 객체를 생성하고 해당 객체를 반환한다.</p>

4. 시스템 테스트

4.1 옵저버 패턴(Observer Pattern) 테스트

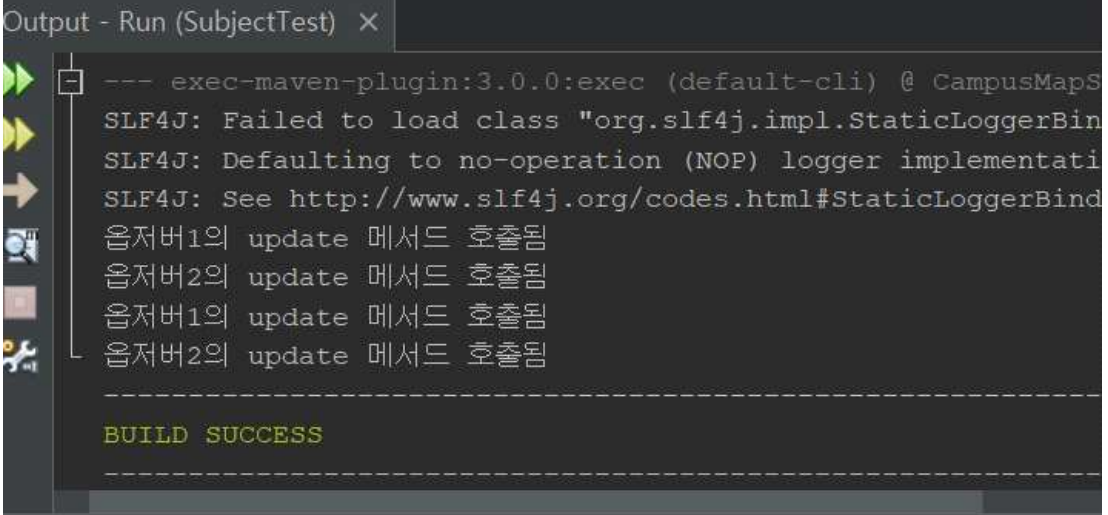
4.1.1 관련 테스트 코드

SubjectTest.java	
중요 코드	<pre>public class SubjectTest { public static void main(String[] args) { BoardData boardData = new BoardData(); LoginData loginData = new LoginData(); new ObserverTest(loginData, boardData); new ObserverTest2(loginData, boardData); boardData.setStatus(null); loginData.setStatus(null); } }</pre>
코드 설명	서브젝트를 구현한 주제객체인 boardData와 loginData를 만들고 옵저버를 구현한 객체에게 해당 주제객체를 생성자로 넘겨주어 옵저버로 등록할 수 있게 하였다. 각각의 주제객체에 대하여 setStatus 메서드를 호출하여 옵저버 객체의 update가 호출될 수 있게 하였다.

ObserverTest.java	
중요 코드	<pre>public class ObserverTest implements Observer{ public ObserverTest(LoginData loginData, BoardData boardData) { loginData.registerObserver(this); boardData.registerObserver(this); } @Override public void update() { System.out.println("옵저버1의 update 메서드 호출됨"); } }</pre>
코드 설명	옵저버 인터페이스를 구현하고 생성자로 주제객체들을 받아와 자신을 옵저버로 등록한다. update를 구현하여 옵저버가 호출되면 update안의 코드가 실행되게 한다.

ObserverTest2.java	
중요 코드	<pre>public class ObserverTest2 implements Observer { public ObserverTest2(LoginData loginData, BoardData boardData) { loginData.registerObserver(this); boardData.registerObserver(this); } @Override public void update() { System.out.println("옵저버2의 update 메서드 호출됨"); } }</pre>
코드 설명	옵저버 인터페이스를 구현하고 생성자로 주제객체들을 받아와 자신을 옵저버로 등록한다. update를 구현하여 옵저버가 호출되면 update안의 코드가 실행되게 한다.

4.1.2 테스트 실행 결과

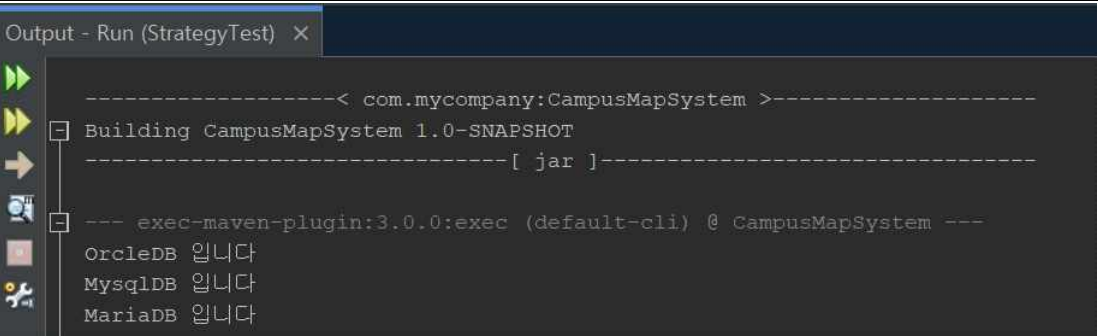
<p>실행 결과</p>	
<p>설명</p>	<p>SubjectTest 객체의 loginData와 BoardData의 setStatus가 실행되어 Observer 인터페이스를 구현한 ObserverTest1과 ObserverTest2 옵저버에서 구현한 update 메서드가 호출되었다.</p>

4.2 전략 패턴(Strategy Pattern) 테스트

4.2.1 관련 테스트 코드

StartegyTest.java	
중요 코드	<pre>public class StrategyTest { public static void main(String[] args) { connectTest(new ConnectOrcleDb()); connectTest(new ConnectMysqlDb()); connectTest(new ConnectMariaDB()); } public static void connectTest(ConnectDB connectDb){ connectDb.getConn(); } }</pre>
코드 설명	connectTest 메서드에 ConnectDB 타입의 인자로 각각 다른 DB 커넥트 객체를 받아와 공통 메서드인 getConn을 실행하였다.

4.2.2 테스트 실행 결과

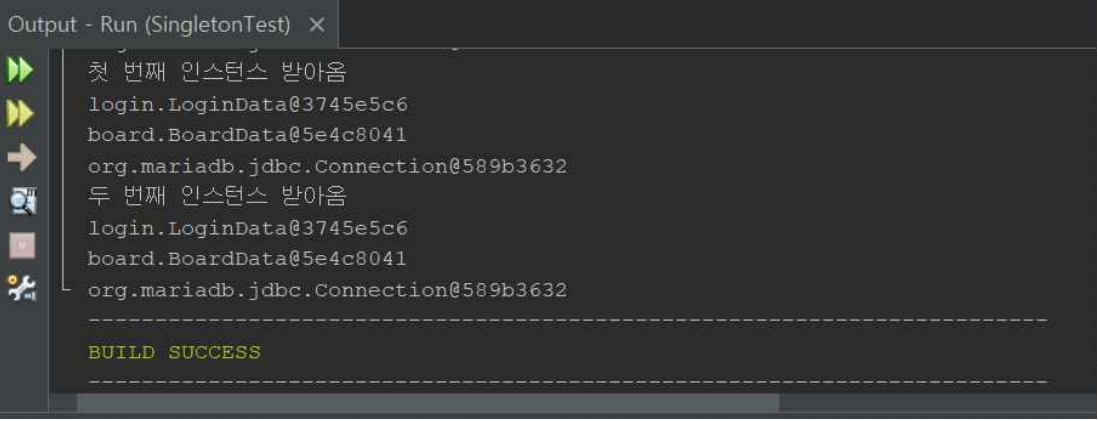
실행 결과	
설명	같은 로직을 수행하였지만 전달된 인자별로 다른 결과가 출력되었다.

4.3 싱글턴 패턴(Singleton Pattern) 테스트

4.3.1 관련 테스트 코드

StartegyTest.java	
중요 코드	<pre> public class SingletonTest { public static void main(String[] args) { System.out.println("첫 번째 인스턴스 받아옴"); System.out.println(DataPool.getInstance().getLoginData()); System.out.println(DataPool.getInstance().getBoardData()); System.out.println(ConnectionPool.getInstance(new ConnectMariaDB()).getConnection()); System.out.println("두 번째 인스턴스 받아옴"); System.out.println(DataPool.getInstance().getLoginData()); System.out.println(DataPool.getInstance().getBoardData()); System.out.println(ConnectionPool.getInstance(new ConnectMariaDB()).getConnection()); } } </pre>
코드 설명	싱글턴 패턴으로 구현되어있는 DataPool과 ConnectionPool 에 정의되어있는 getInstance 메서드를 이용해 싱글턴으로 관리되고 있는 인스턴스들을 받아왔다.

4.3.2 테스트 실행 결과

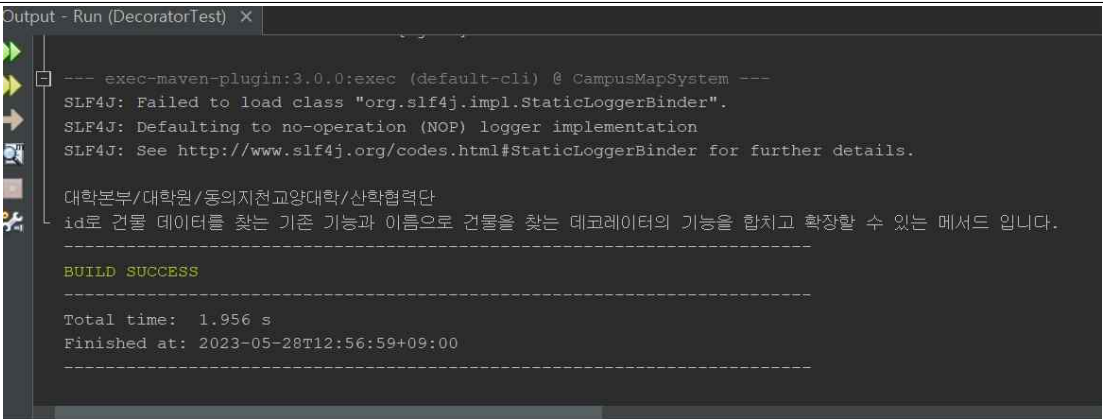
실행 결과	 <pre> Output - Run (SingletonTest) x 첫 번째 인스턴스 받아옴 login.LoginData@3745e5c6 board.BoardData@5e4c8041 org.mariadb.jdbc.Connection@589b3632 두 번째 인스턴스 받아옴 login.LoginData@3745e5c6 board.BoardData@5e4c8041 org.mariadb.jdbc.Connection@589b3632 ----- BUILD SUCCESS ----- </pre>
설명	인스턴스를 두 번 받아왔지만 모두 같은 주소가 리턴되었다.

4.4 데코레이터 패턴(Decorator Pattern) 테스트

4.4.1 관련 테스트 코드

TestBuildingDecorator.java	
중요 코드	<pre> public class TestBuildingDecorator extends DAODecorator<Building, Integer> { public TestBuildingDecorator(DAO dao) { super(dao); } public void testMethod(int n){ Building budilding; System.out.println(); FindByNameBuildingDAO dDao = (FindByNameBuildingDAO)dao; budilding = dDao.findByName(dao.findById(n).getBuName()); System.out.println(budilding.getBuExplain()); System.out.println("id로 건물 데이터를 찾는 기존 기능과 이름으로 건물 을 찾는 데코레이터의 기능을 합치고 확장할 수 있는 메서드 입니다."); } } </pre>
코드 설명	데코레이터 추상 클래스인 DAODecorator를 상속받은 데코레이터 클래스를 추가하였으며 해당 데코레이터에서 기존 UserDAO의 id로 건물 데이터를 기능을 사용하여 건물 객체를 리턴받고 리턴받은 객체의 건물 이름을 이용해 FindByNameBuildingDAO의 기능도 사용하여 다시 한 번 건물의 데이터를 리턴받아 해당 건물 정보를 출력하는 메서드이다.
DecoratorTest.java	
중요 코드	<pre> public class DecoratorTest { public static void main(String[] args) { TestBuildingDecorator test = new TestBuildingDecorator(new FindByNameBuildingDAO(new BuildingDAO())); test.testMethod(1); } } </pre>
코드 설명	새로 확장한 TestBuildingDecorator 객체를 이용해 확장된 메서드를 실행하였다.

4.4.2 테스트 실행 결과

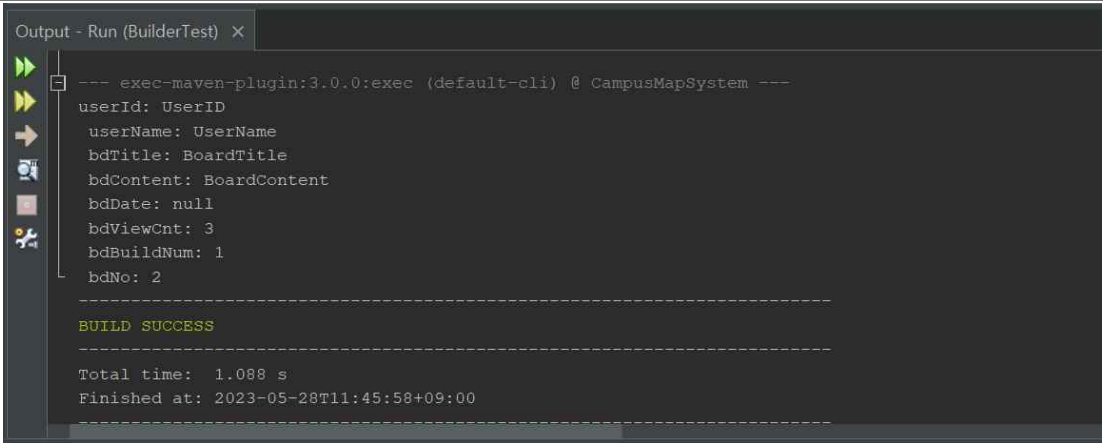
실행 결과	 <pre> Output - Run (DecoratorTest) X --- exec-maven-plugin:3.0.0:exec (default-cli) @ CampusMapSystem --- SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder". SLF4J: Defaulting to no-operation (NOP) logger implementation SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details. 대학본부/대학원/동의지천교양대학/산학협력단 id로 건물 데이터를 찾는 기존 기능과 이름으로 건물을 찾는 데코레이터의 기능을 합치고 확장할 수 있는 메서드 입니다. ----- BUILD SUCCESS ----- Total time: 1.956 s Finished at: 2023-05-28T12:56:59+09:00 ----- </pre>
설명	TestBuildingDecorator의 확장된 기능, FindByNameBuildingDAO의 확장된 기능, BuildingDAO의 기존 기능 모두 정상적으로 실행되었다.

4.5 빌더 패턴(Builder Pattern) 테스트

4.5.1 관련 테스트 코드

BuilderTest.java	
중요 코드	<pre>public class BuilderTest { public static void main(String[] args) { Board board = new BoardBuilder("UserID", "UserName", "BoardTitle", "BoardContent"). bdBuildNum(1). bdNo(2). bdViewCnt(3). build(); System.out.println(board); } }</pre>
코드 설명	BoardBuilder 클래스를 이용하여 필수 속성인 UserID, UserName, BoardTitle, BoardContent를 인자로 넘겨주어 빌더를 생성하고 추가 속성들도 메서드로 설정하여 마지막에 build() 메서드를 호출해 Board 객체를 생성하고 리턴한 후 board객체를 출력하였다.

4.5.2 테스트 실행 결과

실행 결과	
설명	BoardBuilder 가 정상적으로 Build 객체를 생성하고 build객체 정보를 출력하였다.

5. 프로젝트 결과

5.1 프로젝트 완성도

구분	계획 설계 패턴	구현 설계 패턴	완성도
옵저버 패턴	처음 계획할 때 사용자 평가 기능과 주차장 정보 알림 기능에 구현하려고 했지만, 적절하지 않다고 판단함.	사용자의 상태가 바뀌면 바뀐 상태를 저장하고 Observer들에게 알려 사용자의 권한을 변경한다.	100 %
전략 패턴	처음 계획할 때 사용자 정의 기능에 구현하려고 했지만, 적절하지 않다고 판단함.	DAO객체를 만들 때 어떤 DB에 연결할 지 선택, DB를 전략 클래스로 만들어놓고 생성자 인자만 변경해주면, 로직은 변경하지 않고 전략 인터페이스인 ConnectDB만 바꿔주면 연결된 DB만 바뀐다.	100 %
데코레이터 패턴	처음 계획할 때 건물 안내 기능에 구현하려고 했지만, 적절하지 않다고 판단함.	기존 DAO의 기능을 수정하지 않고, 입력받은 건물 이름을 사용하여 해당 건물 정보를 가져오는 로직을 추가하였다.	100 %
빌더 패턴	건물 안내 기능에 건물 안내 객체의 생성 과정을 추상화하여 건물 안내 기능의 유연한 객체 생성을 구현한다.	id, 이름, 제목, 내용을 필수 속성으로 설정하고 메서드를 호출하여 설정된 속성을 바탕으로 객체를 생성하고 반환한다.	100 %
싱글톤 패턴	로그인 기능의 로그인 정보를 담은 객체를 하나만 생성하여 구현하려고 했지만, 적절하지 않다고 판단함.	싱글톤 인스턴스를 사용하여 로그인 데이터에 접근하며, 로그인 기능은 언제나 동일한 인스턴스를 사용한다.	100 %

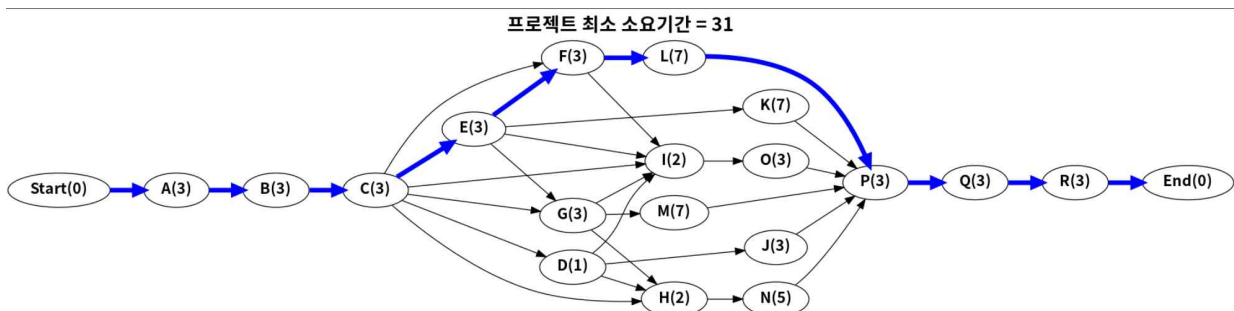
5.2 일정 계획 평가

5.2.1 WBS

Activity No.	이름	소요기간	선행작업	담당자	산출물
Start	시작	0	-	None	
A	요구사항 수집	3	Start	TBD	
B	계획서 작성	3	A	TBD	
C	시스템 설계	3	B	TBD	
D	로그인 기능 설계	1	C	TBD	
E	건물 검색 기능 설계	3	C	TBD	
F	경로 안내 기능 설계	3	C,E	TBD	
G	건물 안내 기능 설계	3	C,E	TBD	
H	사용자 평가 기능 설계	2	C,D,G	TBD	
I	사용자 정의 기능 설계	2	C,D,E,F,G	TBD	
J	로그인 기능 구현	3	D	TBD	
K	건물 검색 기능 구현	7	E	TBD	
L	경로 안내 기능 구현	7	F	TBD	
M	건물 안내 기능 구현	7	G	TBD	
N	사용자 평가 기능 구현	5	H	TBD	
O	사용자 정의 기능 구현	3	I	TBD	
P	시스템 테스트	3	J,K,L,M,N,O	TBD	
Q	시스템 버그 수정	3	P	TBD	
R	최종 보고서 작성	3	Q	TBD	
End	종료	0	R	None	

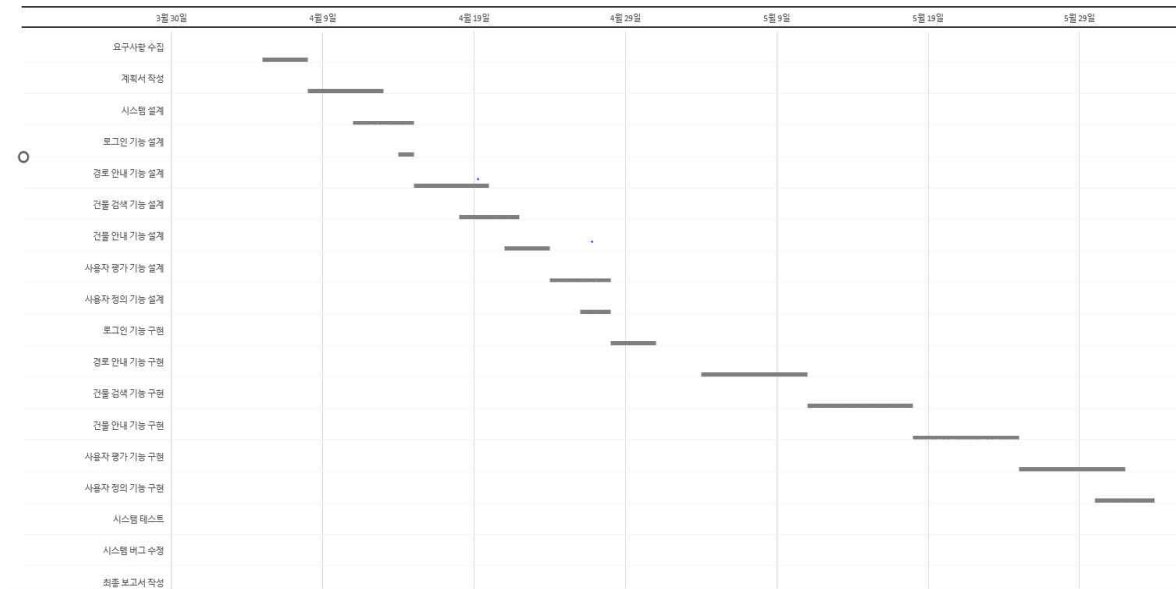
프로젝트 계획 단계에서 작성했던 소작업 목록으로, WBS 작업 분해도와 함께 전체 SW의 설계 및 구현에 있어서 필요한 작업 목록들을 세분화하여, 전체 일정을 고려한 소요기간을 산정하고 그에 맞추어 일정을 진행하였다.

5.2.2 CPM 네트워크



소작업 분해도를 기반으로 설계 당시 예측했던 CPM 네트워크를 도식화하여 임계 경로를 표시한 그림이다. 이를 통하여 전체 프로젝트 수행에 있어서 소요기간 및 중요도에 대해서 한눈에 식별하며 관리를 할 수 있었다.

5.2.3 간트 차트



소작업 분해도를 기반으로 계획서에 기재했던 WBS Gantt Chart이다. 전체 프로젝트의 일정을 한눈에 알아보는데 중점을 두어 각각의 소작업 내용을 수행하는데 초점을 맞추어 프로젝트 진행기간 동안에 작업을 수행할 수 있었다.

5.2.4 평가

캠퍼스 맵 시스템 프로젝트는 사용자들의 위치 찾기와 이동을 원활하게 돕기 위해 설계되었다. 원래의 계획에는 로그인, 위치 안내, 검색, 건물 안내, 사용자 평가, 즐겨찾기 및 주차장 정보 제공 등 다양한 기능들이 포함되어 있었다. 그러나 일정 계획의 변경 및 기술적 제약으로 인해 즐겨찾기 기능과 주차장 정보 제공 기능이 구현되지 못했다.

프로젝트의 목표는 사용자들이 건물 및 시설 위치에 대한 정보를 손쉽게 얻을 수 있도록 지원하는 것이었다. 이 목적은 로그인, 위치 안내, 검색, 건물 안내 및 사용자 평가 기능을 통해 충분히 달성되었다. 전체 시스템을 고려할 때, 즐겨찾기 기능과 주차장 정보 제공 기능은 중요도가 다소 낮은 편이다.

미래에 즐겨찾기 기능과 주차장 정보 제공 기능을 구현할 계획이며, 완성되면 이 기능들이 시스템 전체에서 어떻게 개선될 것인지 기대하고 있다. 이를 통해 사용자들이 보다 편리하게 원하는 건물 및 시설을 찾을 수 있게 되어 전반적인 사용자 경험을 높일 것으로 기대된다.

5.3 형상 관리 평가



형상 관리는 소프트웨어 개발 프로세스에서 핵심적인 역할을 수행하며, 프로젝트의 진행 상황을 효과적으로 추적하고 관리할 수 있게 한다. 우리 팀은 형상관리 도구로써 GitHub를 사용하였고 그 장점을 활용하여 프로젝트의 성공적인 진행을 도모하였다. GitHub의 Git 기반의 분산 버전 관리 시스템을 사용함으로써, 프로젝트에서 중요한 변경 사항들이 체계적으로 기록되어 이전 버전으로 쉽게 되돌아갈 수 있는 높은 안정성과 개발 과정에서의 효율성을 달성할 수 있었다.

GitHub를 사용하여 우리 팀은 소스 코드를 저장하고 이력을 추적하는 데 사용하였고, 팀원들이 소통하며 협업하는 과정이 원활해졌다. GitHub와 연계된 CI/CD 도구를 활용하여 개발 및 배포 프로세스를 자동화하고 지속적 통합을 유지함으로써, 프로젝트 전반의 개발 효율성과 신뢰성을 높일 수 있었다.

5.4 설계 구성요소

항목	내 용
계획	<ul style="list-style-type: none"> ● 각 팀원은 프로젝트 진행에 필요한 모든 작업과 역할을 나열함. ● 팀원 간에 역할을 분배하여 각자의 책임 범위와 업무를 명확히 함. ● Notion과 Discord를 통해 회의 진행, 내용을 요약 정리함. ● 팀원들 간의 회의를 통해 주제에 맞는 기능들을 선정함. ● 선정된 기능들에 대한 소작업을 분류하여 중요도를 설정함. ● 소작업들의 예상 작업시간을 예측하여 CPM을 작성함. ● 주제를 선정하며 캠퍼스 맵에 대한 이해도를 높이고, 스토리와 필요 기능, 필요 데이터 베이스를 준비함.
분석	<ul style="list-style-type: none"> ● 2.1 유스케이스: 회원가입 기능을 수행하기 위한 이벤트 흐름을 기술하였음. ● 2.2 유스케이스: 로그인 기능을 수행하기 위한 이벤트 흐름을 기술하였음. ● 2.3 유스케이스: 건물 검색 기능을 수행하기 위한 이벤트 흐름을 기술하였음. ● 2.4 유스케이스: 건물 안내 기능을 수행하기 위한 이벤트 흐름을 기술하였음. ● 2.5 유스케이스: 게시판 기능을 수행하기 위해 2.5.1 게시물 작성, 2.5.2 게시물 게시물 조회·수정·삭제의 2가지로 세분화하여 각각 이벤트 흐름을 기술하였음.
상세 설계	<ul style="list-style-type: none"> ● 유스케이스에 따라 대략적인 클래스 다이어그램을 그려보고 발생할 문제 상황을 예측하여 그에 적용 가능한 패턴을 의논함. ● 구현을 하다보니 더 세세한 유스케이스가 필요하여 회원가입 유스케이스, 로그인 유스케이스, 건물 검색 유스케이스, 건물 안내 유스케이스, 게시판 기능 유스케이스로 세분화하였고 이에 따른 이벤트 흐름을 작성함. ● 옵저버 패턴: 새로운 다양한 프레임 간의 동기화가 이루어지지 않아 일관성과 데이터 정확성의 문제점을 해결하기 위하여 적용함. ● 전략 패턴: 데이터베이스 변경, 추가 또는 제거 시 코드를 수정해야 하는 문제가 발생할 수 있어 코드의 유연성이 부족하며 각각의 클래스에서 데이터베이스 연결 코드가 중복될 수 있는 문제점을 해결하기 위하여 적용함. ● 싱글톤 패턴: 데이터를 사용하는 모든 유스케이스에서 데이터베이스 커넥션 객체와 옵저버 패턴의 주제 객체인 현재 사용자 데이터 객체, 게시물 데이터 객체 등이 중복 생성되거나 제한된 객체 접근에 문제를 해결하기 위해 적용함. ● 데코레이터 패턴: 기능 추가나 변경이 필요할 때마다 이를 포함하는 클래스를 생성할 때 중복되거나 재사용이 어려운점을 해결하기 위해 적용함. ● 빌더 패턴: 게시판 기능에서 게시물의 정보를 저장하는 객체의 속성이 많을 경우, 각 속성에 대한 생성자를 작성하고 관리하는 것이 복잡해지는 점을 해결하기 위해 적용함.
테스팅 (시험)	<ul style="list-style-type: none"> ● 옵저버 패턴을 구현했음을 증명하기 위하여 시스템 테스트(결과 보고서 4.1.2절)을 수행하였음 ● 전략 패턴을 구현했음을 증명하기 위하여 시스템 테스트(결과 보고서 4.2.2절)을 수행하였음 ● 싱글톤 패턴을 구현했음을 증명하기 위하여 시스템 테스트(결과 보고서 4.3.2절)을 수행하였음 ● 데코레이터 패턴을 구현했음을 증명하기 위하여 시스템 테스트(결과 보고서 4.4.2절)을 수행하였음 ● 빌더 패턴을 구현했음을 증명하기 위하여 시스템 테스트(결과 보고서 4.5.2절)을 수행하였음

5.5 현실적 제한조건

항목	내 용
생산성	<ul style="list-style-type: none"> ● bouml을 이용하여 프로그램을 설계하고, UML을 이용하여 클래스 다이어그램을 쉽게 그리고, 프로그램을 구현 할 때 짧은 시간으로 클래스를 빠르게 구현하였다. ● 형상관리 도구로 Git을 사용하여 개발자들 간에 코드를 쉽고 직관적으로 공유함으로써, 개발자 간의 의견 공유가 쉬워져 더 좋은 프로그램을 생산할 수 있으며, 유지 보수가 쉬워진다. ● 기존의 존재하던 동의대학교 캠퍼스에 대한 정보를 보다 쉽게 사용자에게 제공, 안내할 수 있고, 찾는 시간도 단축할 수 있다. ● 기존의 클래스만을 대상으로 하는 프로그래밍의 한계를 극복하고 여러 소프트웨어 설계 패턴들을 적용하여 필요하지 않은 코딩 작업을 줄이고, 보다 좋은 구조를 가지는 프로그램을 제작하여 소프트웨어 유지 보수를 원활하게 한다.
산업표준	<ul style="list-style-type: none"> ● 객체지향 분석 및 설계의 기본 요소인 클래스 표기 방법인 UML 클래스 다이어그램을 활용하였다. ● 개발 프로세스로 애자일 개발 방법론인 스크럼을 사용하여 프로젝트의 요구사항 변화에 대응하기 쉬웠고, 빠르게 피드백을 반영하고 문제점을 개선하였으며, 팀워크 강화, 책임감 증대, 높은 품질의 결과물을 얻을 수 있었다. ● 개발 단계에서 테스트시 ISO/IEC/IEEE 29119 SW 테스트 국제 표준에서 정의하는 테스트 케이스 설계 기법(경계값 분석, 동등 분할, 문장 테스트, 분기 테스트 등)을 적용하였다. ● Git을 이용한 형상관리를 하여 소프트웨어의 유지보수를 원활하게 하여 사용자에게 더 좋은 서비스를 제공하였다.
기타	<ul style="list-style-type: none"> ● NetBeans IDE의 maven 빌드 도구를 사용하여 라이브러리 설치를 자동으로 하게 하였으며, FindBugs 정적 분석 도구를 사용하여 발견된 주요 결함(major 이상)을 제거하였다. ● 팀원 각각의 시간 맞추기가 힘들어 Notion, Discode 등을 이용한 비대면 회의를 진행한다.

6. 소감

이름	소감
김민곤	<p>캠퍼스 맵 시스템 프로젝트를 진행하면서 팀원들과 협력하여 다양한 기능과 디자인 패턴을 적용해보는 소중한 경험을 얻었습니다. 프로젝트가 시작될 때부터 목표로 했던 사용자 중심의 기능 구현을 완성할 수 있어 뿌듯함을 느꼈습니다.</p> <p>이 프로젝트를 통해 기술적인 능력뿐만 아니라 프로젝트 관리, 의사소통, 문제 해결 능력 등 소프트 스킬들을 향상시킬 수 있는 좋은 기회였습니다. 특히, 다양한 도메인의 학습자료를 통해 새로운 지식을 습득하고 그것들을 실제 프로젝트에 적용할 수 있는 능력을 키울 수 있었습니다.</p> <p>결론적으로, 동의대학교 캠퍼스 맵 시스템 프로젝트는 전공 기술 역량을 강화하고 전체적인 소프트웨어 개발 역량을 높일 수 있는 알찬 시간이었습니다. 앞으로 더 나아가 다양한 프로젝트에서 이 경험들을 살려 성장하는 개발자가 되겠습니다.</p>
박준석	<p>소프트웨어 설계 공학 프로젝트를 진행하며 자신의 부족함을 느낄 수 있었습니다. 이번 프로젝트를 진행하며 팀원들의 개인 사정 또는 외부 사정으로 인한 변수로 처음 계획서와는 많은 부분이 다르게 진행되었습니다. 무엇보다, 직전 학기에는 프로젝트의 주제를 선정하고 해당 기능을 구현하기만 하면 되었기에 큰 어려움을 느끼지 못했습니다. 하지만, 적절한 디자인 패턴을 찾고 적용하는 과정에서 생각보다 많은 어려움을 겪었습니다. 억지로 패턴을 사용하려다 이후 회의에서 패턴이 적절하지 않다고 판단되어 시간이 지체되는 경우가 잦았습니다. 다행히 팀원들이 열심히 팀프로젝트에 참여하여 디자인 패턴을 모두 적용하여 캠퍼스 맵 프로젝트를 성공적으로 끝낼 수 있었다고 생각합니다.</p> <p>이번 프로젝트를 진행하며 개인적으로 많은 부분을 얻어가는 것 같습니다. 개인적으로 프로그래밍 공부의 부족함을 통감할 수 있었고, 그럼에도 프로젝트를 진행하면서, 스크럼 등을 통해 프로젝트 일정을 관리하는 방법, 디자인 패턴을 적절히 사용하는 것과, 디자인 패턴을 적절히 사용하였을 경우 SW 설계에 있어서 얼마나 큰 이점이 생기는지 또한 알게 되어 성취에 대한 목표가 생긴 것 같습니다. 비록 팀 프로젝트를 진행하며 이런저런 문제점이 있긴 했지만 그럼에도 역량을 높일 수 있는 기회가 되었던 것 같습니다.</p>
한정우	<p>이번 캠퍼스 맵 프로젝트는 우리 팀에게 소중한 경험을 안겨준 과정이었습니다. 프로젝트를 진행하며 개인 뿐만 아니라 팀 전체로 성장할 수 있는 기회였고, 그 결과로 탄생한 캠퍼스 맵 앱은 사용자들에게도 도움이 중요한 시스템이 되었다고 생각합니다.</p> <p>느낀점은 현실적인 문제와 과제들을 직면하며 프로젝트 관리와 진행을 배웠습니다. 발전된 부분은 필요한 개선 사항들을 지속적으로 확인하며 최적화된 결과물을 만들어가는 과정에서 발전한 것 같습니다. 일정과 기술적 제약으로 인해 몇 가지 기능을 완성하지 못했던 점이 아쉬웠습니다. 팀원들 간에 서로 존중하고 협력하는 분위기를 유지하여 소통 및 작업을 원활하게 진행할 수 있어서 좋았습니다. 총체적으로 볼 때, 캠퍼스 맵 프로젝트는 우리 팀 모두에게 큰 의미가 있는 경험이었습니다. 이를 통해 개인적으로도 발전하고, 또한 팀의 역량을 높여 나갈 수 있었습니다.</p>