

---

소프트웨어 공학

[ Issue Tracking System of IssueStation ]

---



과목명	소프트웨어 공학
교수명	이찬근 교수님
제출일	2024. 06.03
팀 명	이슈역
팀 원 / 학 번 / 역할	한동희 / 20210189 / Back-end 민보욱 / 20215744 / Back-end 전은진 / 20210117 / Front-end 이동훈 / 20216018 / Front-end 박상훈 / 20216942 / project 문서화

## < 목 차 >

### 1. 프로젝트 설명

- 1.1. '이슈 관리 시스템' 설명
- 1.2. 요구 명세서

### 2. 요구 정의 및 분석

- 2.1. Use Case Diagram
  - 2.1.1. 액터
  - 2.1.2. 유스케이스
  - 2.1.3. 유스케이스 간의 관계
- 2.2. Use Case Specification
- 2.3. Domain Model
- 2.4. SSD

### 3. 설계

- 3.1. Diagram
  - 3.1.1. Class diagram
- 3.2. OOAD/GRASP 패턴/설계 원칙
- 3.3. 추가적인 UML Diagram
  - 3.3.1. Entity Relationship diagram

### 4. 구현

- 4.1. 주요 기능 Screen shot
- 4.2. 주요 기능 설명
- 4.3 API 기능 명세

### 5. Test Case 수행 내역

- 5.1. Test Case 기본 기능 및 추가 기능
  - 5.1.1. Test Case 결과

### 6. GitHub Project 활용

- 6.1 GitHub Project Address of IssueStation
- 6.2. Project Progress history Screen shot
- 6.3 팀원 역할

### 7. Video Clip

## 1. 프로젝트 설명

### 1.1. 이슈 관리 시스템

팀 이슈역의 프로젝트는 주어진 조건들과 추가적 옵션들을 가지고 이슈 관리 시스템을 개발하는 것이다. 이슈 관리 시스템은 이슈 목록과 내용등을 체계적으로 관리하는 소프트웨어이다. 본 프로젝트에서는 이슈 등록 및 관리와 같은 시스템의 전형적인 기능을 포함할 것이고, MUST 포함 기능과 추가 기능은 다음과 같이 있다.

- 계정 추가
- 이슈 배정을 포함한 이슈 상태 변경
- 이슈 브라우즈 및 검색
- 이슈 통계 분석
- 이슈 등록
- Assignee 자동 추천 기능 (추가 기능)
- 이슈 코멘트 추가
- Private/Public으로 접근성 관리 (추가 기능)
- 이슈 상세 정보 확인
- GUI (추가 기능)
- 다중 UI (추가 기능)

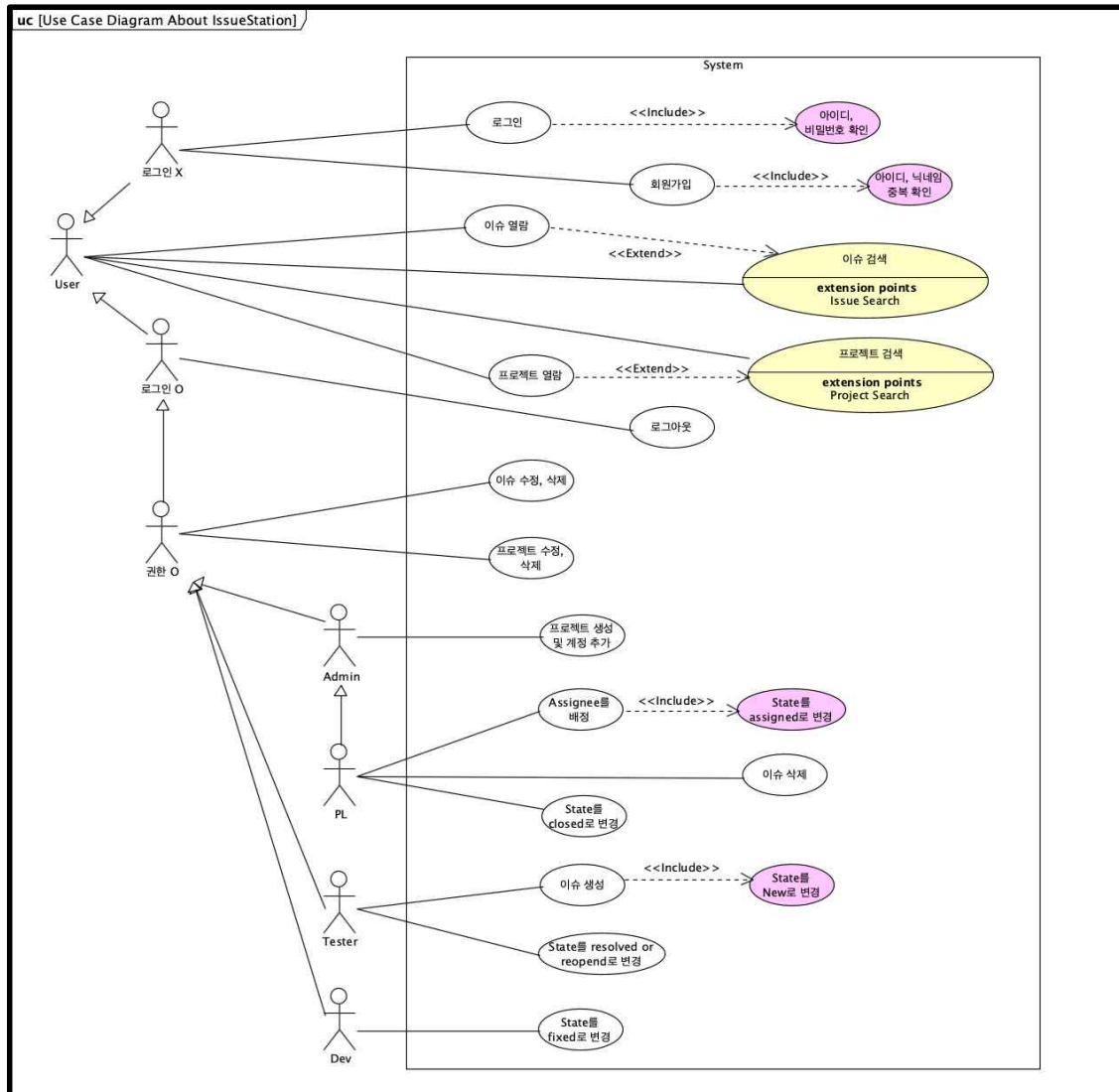
※ 프로젝트에서 다음과 같은 기능들이 추가 됐다. “Private/Public으로 접근성 관리”, “GUI”, “다중 UI”, “Assignee 자동 추천 기능”

### 1.2. 요구 명세서

기능 요구사항	비기능 요구사항
<ul style="list-style-type: none"><li>i) 이슈를 생성할 수 있어야 한다.</li><li>ii) 이슈 생성 시 제목, 상태, 담당자 등을 입력할 수 있어야 한다.</li><li>iii) 생성된 이슈는 기본적으로 'New'상태로 설정되어야 한다.</li><li>iv) 기존 이슈의 정보를 수정할 수 있어야한다.</li><li>v) 수정 가능한 항목에는 상태, 담당자등이 포함된다.</li><li>vi) 이슈의 상태는 'New', 'Assigned', 'Resolved', 'Reopened', 'Fixed', 'Closed'등으로 변경할 수 있어야 한다.</li><li>vii) 이슈 세부 정보를 열람할 수 있어야 한다.</li><li>ix) 코멘트를 추가할 수 있어야 한다.</li></ul>	<ul style="list-style-type: none"><li>i) 시스템은 여러명의 사용자가 접속해도 원활하게 동작해야 한다.</li><li>ii) 사용자는 역할에 따라 접근할 수 있는 기능이 제한되어야 한다. 예를 들어, 일반 사용자는 이슈를 검색하고 열람할 수 있지만, 관리자만 이슈 생성, 삭제등을 할 수 있다.</li><li>iii) 시스템은 직관적이고 사용하기 쉬운 인터페이스를 제공해야 한다.</li><li>iv) 시스템은 웹 기반으로 개발되어야 한다.</li><li>v) 데이터 베이스는 MySQL을 사용해야 한다.</li><li>VI) 일정 시간마다 토큰을 새로 발행해야 한다.</li></ul>

## 2. 요구 정의 및 분석

### 2.1. Use Case Diagram



<Figure1. Use Case Diagram>

#### 2.1.1. 액터(Actor)

- 1) 사용자(User): 이슈 열람과 프로젝트 열람을 하는 일반 사용자이다. 사용자는 상황에 따라 이슈 검색과 프로젝트 검색을 할 수 있다.
- 2) 로그인 X: 이 액터는 User를 상속 받으며 로그인과 회원가입을 하는 액터이다.
- 3) 로그인 O: 이 액터는 User를 상속 받으며 로그아웃을 수행할 수 있다.

- 4) 권한 O: 로그인 O 액터를 상속 받고, 이슈 수정 및 삭제, 프로젝트 수정 및 삭제를 수행할 수 있는 권한을 가졌다.
- 5) 관리자(Admin): 권한 O 액터를 상속 받았으며 프로젝트를 생성 하고, 계정 추가를 할 수 있다.
- 6) 프로젝트 리더(PL): Admin을 상속 받았다. PL 액터는 이슈를 삭제할 수 있으며 assignee 배정과 이슈의 상태를 closed를 바꾼다.
- 7) 테스터(Tester): 로그인 O 액터를 상속 받았다. 이슈를 생성할 수 있으며 이슈의 상태를 resolved 또는 reopend로 바꾼다.
- 8) 개발자(Dev): 로그인 O 액터를 상속 받았다. 이슈의 상태를 fiexd로 바꾼다.

### 2.1.2. 유스케이스(Use Case Diagram)

- 1) 로그인: 사용자가 ID, PW를 입력하여 로그인 한다.
- 2) 로그아웃: 사용자가 시스템에서 로그아웃하여 세션을 종료한다.
- 3) 아이디, 비밀번호 일치 확인: 시스템이 입력된 ID와 PW가 데이터베이스에 저장된 정보와 일치하는지 확인한다.
- 4) 회원가입: 아직 회원이 아닌 사용자가 시스템에 정보를 입력하여 회원으로 등록한다.
- 5) 아이디, 닉네임 중복 확인: 회원가입 시 입력한 ID와 닉네임이 기존의 회원 정보와 중복되지 않는지 시스템이 확인한다.
- 6) 이슈 열람: 사용자가 등록된 이슈들을 조회한다.
- 7) 이슈 검색: 사용자가 특정 키워드로 이슈들을 검색한다.
- 8) 프로젝트 열람: 사용자가 프로젝트들을 조회한다.
- 9) 프로젝트 검색: 사용자가 특정 키워드로 프로젝트를 검색한다.
- 10) 이슈 수정/삭제: 등록된 이슈의 내용을 수정하거나 삭제한다.
- 11) 프로젝트 수정/삭제: 등록된 프로젝트의 내용을 수정하거나 삭제한다.
- 12) 프로젝트 생성/계정 추가: 프로젝트를 생성하거나 기존 프로젝트에 계정을 추가한다.
- 13) Assignee를 배정: 특정 이슈에 대해 담당자를 지정한다.
- 14) state를 assigned로 변경: 이슈의 상태를 'assigned'으로 변경한다.
- 15) 이슈 삭제: 등록된 이슈를 시스템에서 제거한다.
- 16) state를 closed로 변경: 이슈의 상태를 '닫힘(closed)'으로 변경한다.
- 17) 이슈 생성: 새로운 이슈를 시스템에 등록한다.
- 18) state를 New로 변경: 이슈의 상태를 'new'으로 변경한다.
- 19) state를 resolved or reopend로 변경: 이슈의 상태를 'resolved' 또는 'reopened'으로 변경한다.
- 20) state를 fixed로 변경: 이슈의 상태를 'fixed'으로 변경한다.

### 2.1.3. 유스케이스 간의 관계(Use Case Diagram Relationship)

#### 2.1.3.1 <<include>> 관계

- 1) 로그인을 하게 되면 아이디, 비밀번호 확인을 필요로 하기 때문에 [로그인] 유스케이스는 [아이디, 비밀번호 확인]을 선행하기에 <<include>> 관계가 성립된다.
- 2) 회원가입시 아이디, 닉네임 중복 확인 과정이 필요하다. 따라서 [회원가입] 유스케이스는 [아이디, 닉네임 중복 확인]을 선행하기에 <<include>> 관계가 성립된다.
- 3) Assignee가 배정될 때, 이슈의 상태가 'assigned'로 변경되는 것은 필수적인 절차이다. 따라서, [Assignee 배정] 유스케이스와 [State를 assigned로 변경] 유스케이스 사이에는 <<include>> 관계가 성립된다.
- 4) Tester가 이슈를 생성할 때, 해당 이슈의 상태가 'New'로 변경되는 것은 필수적인 절차이다. 이에 따라, [이슈 생성] 유스케이스와 [State를 'New'로 변경] 유스케이스 사이에는 <<include>> 관계가 성립된다.

#### 2.1.3.2 <<extend>> 관계

- 1) 이슈를 검색할 때, 검색된 이슈를 열람할 수 있는 기능이 확장 옵션으로 제공된다. 따라서 [이슈 검색] 유스케이스와 [이슈 열람] 유스케이스는 <<extend>> 관계를 가지게 된다.
- 2) 프로젝트를 검색할 때, 프로젝트를 열람하는 기능이 확장 옵션으로 제공될 수 있다. 따라서 [프로젝트 열람] 유스케이스는 [프로젝트 열람] 유스케이스와 <<extend>> 관계를 가지게 된다.

## 2.2. Use Case Diagram Specification

유스케이스 명	액터	시작 조건
프로젝트 생성 및 계정 추가	Admin	Admin이 로그인 되어있어야 한다.
프로젝트 검색	User	
프로젝트 열람	User	User가 열람할 프로젝트가 있어야한다.
프로젝트 삭제	Admin	1.Admin이 원하는 프로젝트에 대한 프로젝트 열람 화면에 접속한다. 2.System은 권한이 있는 Admin에게 "Delete Project"버튼을 표시해준다.
이슈 생성	Tester	1.Tester가 로그인인 상태 2.프로젝트가 존재하여 해당 프로젝트에 Tester 권한이 있어야 한다
이슈 검색	User	1.Private 프로젝트에 대한 이슈 검색 2.액터들이 로그인 되어 있어야 한다. 3.프로젝트에 대한 권한이 있어야 한다. + Public 프로젝트에 대한 이슈 검색은 조건 X
이슈 열람	User	1.Private 프로젝트에 대한 이슈의 정보 열람 2.액터들이 로그인 되어 있어야 한다. 3.프로젝트에 대한 권한이 있어야 한다. + Public 프로젝트에 대한 이슈의 정보 열람은 조건X
이슈 삭제	PL	1.PL이 원하는 이슈에 대한 이슈 열람 화면에 접속 2.System은 권한이 있는 PL에게 "Delete Issue"버튼을 표시해준다.
state를 assigned로 변경	PL	1.PL이 원하는 이슈에 대한 이슈 열람 화면에 접속 2.System은 권한이 있는 PL에게 "Assigned State" 버튼을 표시해준다.
state를 fixed로 변경	Developer	1.Developer가 할당받은 이슈에 대한 이슈 열람 화면에 접속 2.System은 권한이 있는 Developer에게 "Fixed State" 버튼을 표시해준다.
state를 resolved or reopened로 변경	Tester	1.Reporter가 본인인 이슈에 대한 이슈 열람 화면에 접속 2.해당 이슈의 State가 "Fixed"여야 한다. 3.System은 권한이 있는 Tester에게 "Resolved or Reopened"버튼을 표시해준다.
state를 closed로 변경	PL	1.PL이 이슈의 정보 열람 화면에 접속 2.선택한 이슈의 State가 "Resolved"여야 한다. 3.System은 "Closed Issue"버튼을 표시해준다.
회원가입	로그인 X User	System은 "회원가입" 버튼을 표시해준다.
로그인	로그인 X User	System은 "로그인" 버튼을 표시해준다.
로그아웃	로그인 O User	System은 "로그아웃" 버튼을 표시해준다.

기본 흐름	대안 흐름	종료 조건
<p>1.Admin이 프로젝트 생성 버튼을 클릭한다.</p> <p>2.Admin이 각 역할에 대해 프로젝트에 참가하는 계정을 추가한다 (=권한 부여)</p> <p>3.Admin이 프로젝트 열람 옵션을 Public으로 설정한다.</p> <p>4.Admin이 프로젝트 생성 완료 버튼을 클릭한다.</p>	<p>A1. 프로젝트 열람 옵션 Private으로 설정. -기본 시나리오 3에서 분기</p> <p>A1.1 Admin이 Private 옵션으로 설정한다.</p> <p>A1.2 Admin이 프로젝트 생성 완료 버튼을 클릭한다.</p>	<p>“프로젝트가 생성되었습니다!”</p> <p>안내창이 화면에 띄워진다.</p> <p>프로젝트가 화면에 생성된다.</p> <p>생성한 프로젝트 화면으로 리다이렉트</p>
<p>1.“프로젝트 검색” 창에 검색어를 입력한다.</p> <p>2.System이 검색 결과를 화면에 보여준다.</p>		<p>User가 정한 기준에 맞는 검색 결과가 화면에 나온다.</p>
<p>1.‘프로젝트 검색’ 유스케이스를 포함한다.</p> <p>2.Public 프로젝트를 클릭한다.</p> <p>3.System이 클릭한 프로젝트에 대한 화면을 보여준다.</p>	<p>A1. 로그인 X User가 Private 프로젝트 열람 -기본 시나리오 2에서 분기 A1.1 로그인 X User가 Private 프로젝트를 클릭한다. A1.2 System이 로그인 X User에게 로그인이라고 알림창을 띄워준다. A1.3 로그인 화면으로 리다이렉트</p> <p>A2. 로그인 O &amp; 권한 X User가 Private 프로젝트 열람 -기본 시나리오 2에서 분기 A2.1 로그인 O &amp; 권한 X User가 Private 프로젝트를 클릭한다. A2.2 System이 권한이 없다고 알림창을 띄워준다. A3 . 로그인 O &amp; 권한 O User가 Private 프로젝트 열람 -기본 시나리오 2에서 분기 A3.1 로그인 O &amp; 권한 O User가 Private 프로젝트를 클릭한다.</p>	<p>User가 열람할 프로젝트의 내용이 화면에 나온다.</p>
<p>1.Admin은 “Delete Project” 버튼을 클릭한다.</p> <p>2.System은 삭제할 Project의 이름을 타이핑하도록 하는 Input 창을 띄운다.</p> <p>3.Admin은 Project의 이름을 정확히 타이핑하고 “Delete”버튼을 클릭한다.</p>	<p>A1. 타이핑이 정확하지 않음 -기본 시나리오 3에서 분기 A1.1 System은 “Delete” 버튼을 비활성화하고 타이핑이 정확하지 않다는 안내 문구를 표시해준다.</p>	<p>“프로젝트가 삭제되었습니다!” 안내창이 화면에 띄워준다.</p> <p>해당 프로젝트가 화면에서 삭제되어 보여진다.</p>
<p>1.Tester가 이슈 생성 버튼을 클릭한다.</p> <p>2.Tester가 이슈 정보 입력(이슈 이름, 이슈 중요도 등)</p> <p>3.Tester가 코멘트를 추가한다.</p> <p>4.System은 State 상태를 NEW로 변경한다.</p> <p>5.System이 Reporter의 값을 Tester Name으로 설정한다.</p> <p>6.Tester가 이슈 생성 완료 확인 버튼을 클릭한다.</p>	<p>A1. 코멘트 추가 없음 -기본 시나리오 3에서 분기 A1.1 Tester가 코멘트를 추가하지 않는다.(넘어감) A1.2 System은 State 상태를 NEW로 변경 A1.3 System이 Reporter의 값을 Tester Name으로 설정 A1.4 Tester가 이슈 생성 완료 확인 버튼을 클릭</p>	<p>“이슈 생성이 완료되었습니다!” 안내문구 화면에 나온다.</p> <p>이슈가 프로젝트 안에 생성된다.</p> <p>해당 이슈 페이지로 리다이렉트</p>
<p>1.‘프로젝트 검색’ 유스 케이스를 포함한다.</p> <p>2.‘프로젝트 이슈 열람’ 유스 케이스를 포함한다.</p> <p>3.이슈 상태로 이슈를 검색한다.</p> <p>4.System이 검색 결과를 화면에 보여준다.</p>	<p>A1. 키워드로 검색 -기본 시나리오 3에서 분기 A1.1 이슈 이름으로 이슈를 검색한다. A1.2 System이 검색 결과를 화면에 보여준다.</p> <p>A2 중요도로 검색 -기본 시나리오 3에서 분기 A2.1 중요도로 이슈를 검색한다. A2.2 System이 검색 결과를 화면에 보여준다.</p>	<p>User가 정한 기준에 맞는 검색 결과가 화면에 나온다.</p>

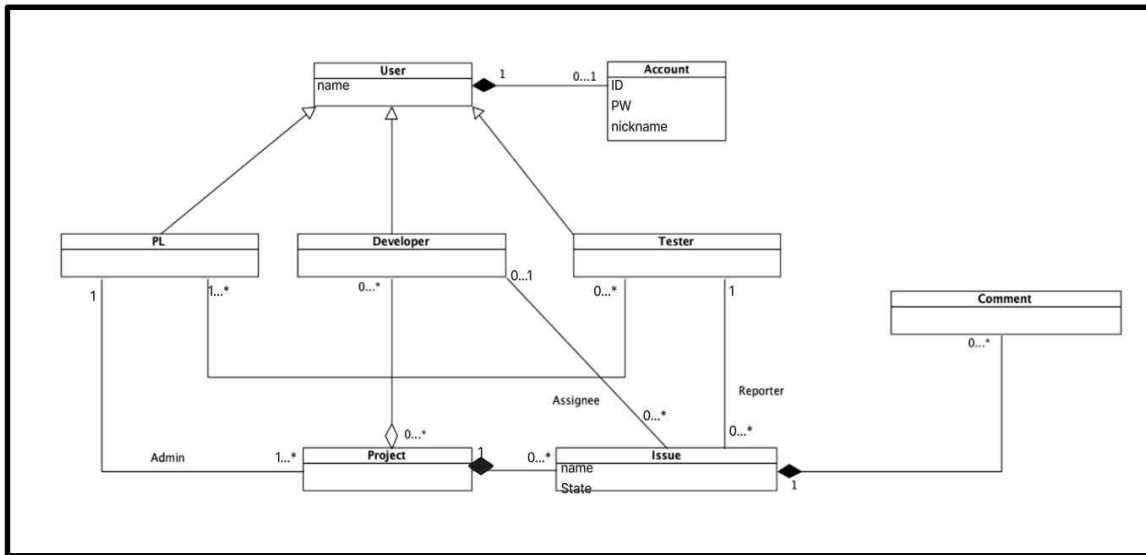


	<p>A3. 배정 여부로 검색 -기본 시나리오 3에서 분기 A3.1 자신(User)이 배정된 이슈를 검색한다. A3.2 System이 검색 결과를 화면에 보여준다.</p> <p>A4. 네 가지 기준 선택적으로 골라서 검색 -기본 시나리오 3에서 분기 A4. 1 이슈 상태, 키워드, 중요도, 배정 여부 중 골라서 이슈 검색 A4.2 System이 검색 결과를 화면에 보여준다.</p>	
<p>1. '이슈 검색' 유스 케이스를 포함한다.</p> <p>2. User가 정보 열람하려는 이슈를 선택한다.</p> <p>3. System이 해당 이슈에 대한 정보를 화면에 보여준다.</p>	<p>A1. 검색없이 바로 선택 A1.1 User가 정보 열람하려는 이슈를 선택한다. A1.2 System이 해당 이슈에 대한 정보를 화면에 보여준다.</p>	User가 선택한 이슈에 맞는 정보가 화면에 나온다.
<p>1. PL은 "Delete Issue" 버튼을 클릭한다.</p> <p>2. System은 삭제할 Issue의 이름을 타이핑하도록 하는 Input창을 띄운다.</p> <p>3. PL은 Issue의 이름을 정확히 타이핑하고 "Delete" 버튼을 클릭한다.</p>	<p>A1. 타이핑이 정확하지 않음 기본 시나리오 3에서 분기 A1.1 System은 "Delete" 버튼을 비활성화하고 타이핑이 정확하지 않다는 안내 문구를 표시해준다.</p>	<p>"이슈가 삭제되었습니다!" 안내문구 띄워준다.</p> <p>프로젝트 안에 해당 이슈가 삭제되어 보여진다.</p>
<p>1. PL은 "Assigned State" 버튼을 클릭한다.</p> <p>2. Tester가 repoter로 되어 있는 해당 이슈에 대해 적절한 개발자를 선택 후 선택한 작업자를 이슈의 assignee로 지정.</p> <p>3. PL이 코멘트를 추가한다.</p> <p>4. PL이 "Assigned Complete" 버튼을 클릭한다.</p> <p>5. System이 이슈 상태를 'assigned'로 변경한다.</p>	<p>A1. Assignee 선택 X -기본 시나리오 2에서 분기 A1.1 System이 "Assigned Complete" 버튼을 비활성화한다. A1.2 tester가 repoter로 되어 있는 해당 이슈에 대해 적절한 개발자를 선택 후 선택한 작업자를 이슈의 assignee로 지정. A1.3 PL이 "Assigned Complete" 버튼을 클릭 A1.4 System이 이슈 상태를 'assigned'로 변경한다.</p> <p>B1. 코멘트 추가 없음 -기본 시나리오 3에서 분기 B1.1 PL이 코멘트를 추가하지 않는다.(넘어감) B1.2 System은 State 상태를 "assigned"로 변경 B1.3 PL이 "Assigned Complete" 버튼을 클릭 B1.4 System이 이슈 상태를 'Assigned'로 변경한다.</p>	<p>"이슈의 상태가 Assigned로 변경되었습니다!" 안내문구 띄워준다.</p> <p>화면에서 이슈 상태가 Assigned로 보여진다.</p>
<p>1. Developer은 "Fixed State" 버튼을 클릭한다.</p> <p>2. Developer가 이슈의 상태를 Assigned에서 Fixed로 변경.</p> <p>3. Developer가 코멘트를 남긴다.</p> <p>4. Developer가 "Fixed Complete" 버튼을 클릭</p>	<p>A1. Assigned Issue A1.1 자신에게 할당된 Issue가 아닌데 열람하려고 하면 "해당 이슈의 담당자가 아닙니다!" 안내 메시지를 띄운다. A1.2 확인 버튼을 누르면 할당된 이슈로 리다이렉트한다.</p> <p>B1. 코멘트 추가 없음 -기본 시나리오 3에서 분기 B1.1 Developer가 코멘트를 추가하지 않는다.(넘어감) B1.2 Developer가 "Fixed Complete" 버튼을 클릭</p>	<p>"이슈의 상태가 Fixed로 변경되었습니다!" 안내문구 띄워준다.</p> <p>화면에서 이슈 상태가 Fixed로 보여진다.</p>
<p>1. Tester은 "Resolved or Reopened" 버튼을 클릭한다.</p> <p>2. Tester가 이슈의 상태를 Fixed에서 Resolved로 변경.</p> <p>3. Tester가 코멘트를 추가한다.</p> <p>4. Tester가 "Complete" 버튼을 클릭</p>	<p>"이슈 상태가 Resolved/Reopened로 변경되었습니다!" 안내문구를 띄워준다.</p> <p>화면에서 이슈 상태가 Resolved/Reopened로 보여진다.</p>	<p>"이슈 상태가 Resolved/Reopened로 변경되었습니다!" 안내문구를 띄워준다.</p> <p>화면에서 이슈 상태가 Resolved/Reopened로 보여진다.</p>

1.PL은 “Closed Issue”버튼을 클릭함 2.이슈의 상태를 Closed로 변경. 3.PL이 “Complete” 버튼을 클릭		“이슈의 상태가 Closed로 변경되었습니다!” 안내문구 띄워준다.  화면에서 이슈 상태가 Closed로 보여진다.
1.로그인X User은 “회원가입” 버튼을 클릭함 2.System은 아이디와 비밀번호, 닉네임을 입력하는 창을 보여줌 3.System은 아이디와 닉네임 중복 체크를 함 4.로그인X User은 “Complete” 버튼을 클릭함	A1. 입력된 정보가 올바르지 않을 때 -기본 시나리오 2에서 분기 A1.1 System은 Complete 버튼을 비활성화한다 A1.2 System은 어느 정보가 올바르지 않은 지 표시해준다	“회원가입이 완료되었습니다!” 안내문구를 띄워준다.  회원 정보가 시스템에 저장된다.
1.로그인X User은 “로그인” 버튼을 클릭한다. 2.System은 아이디와 비밀번호를 입력하는 창을 보여준다. 3.System은 아이디와 닉네임을 확인한다. 4.로그인X User은 “로그인하기” 버튼을 클릭한다.	A1. 아이디와 비밀번호가 일치하지 않을 때 -기본 시나리오 4에서 분기 A1.1 System은 “아이디와 비밀번호가 올바르지 않습니다” 안내문구를 띄워준다.	“로그인이 완료되었습니다!” 안내문구를 띄워준다.  로그인이 된 상태의 화면이 보여진다.
1.로그인O User은 “로그아웃” 버튼을 클릭함 2.System은 “정말 로그아웃 하시겠습니까?” 안내창을 보여준다. 3.로그인O User은 “네” 버튼을 클릭한다. 4.System이 안내창을 끈다.	A1. 로그아웃 “아니요 버튼” -기본 시나리오 3에서 분기 A1.1 로그인 O User가 ‘아니요’ 버튼을 클릭한다. A1.2 System은 안내창을 끈다.	“로그아웃이 완료되었습니다!” 안내문구를 띄워준다  로그아웃된 화면이 보여진다.

- ☑ 이 유스케이스 명세는 유스케이스 명, 액터, 시작 조건, 기본 흐름, 대안 흐름, 종료 조건으로 구성되며 “이슈 관리 시스템”의 핵심 기능과 사용자 상호작용을 정의한다.
- 목적은 사용자가 해당하는 유스케이스 명에 따라 각각의 프로세스를 명확히 하는 것이고, 주 참여자는 PL, Developer, Tester 그리고 Admin이다.

## 2.3. Domain model



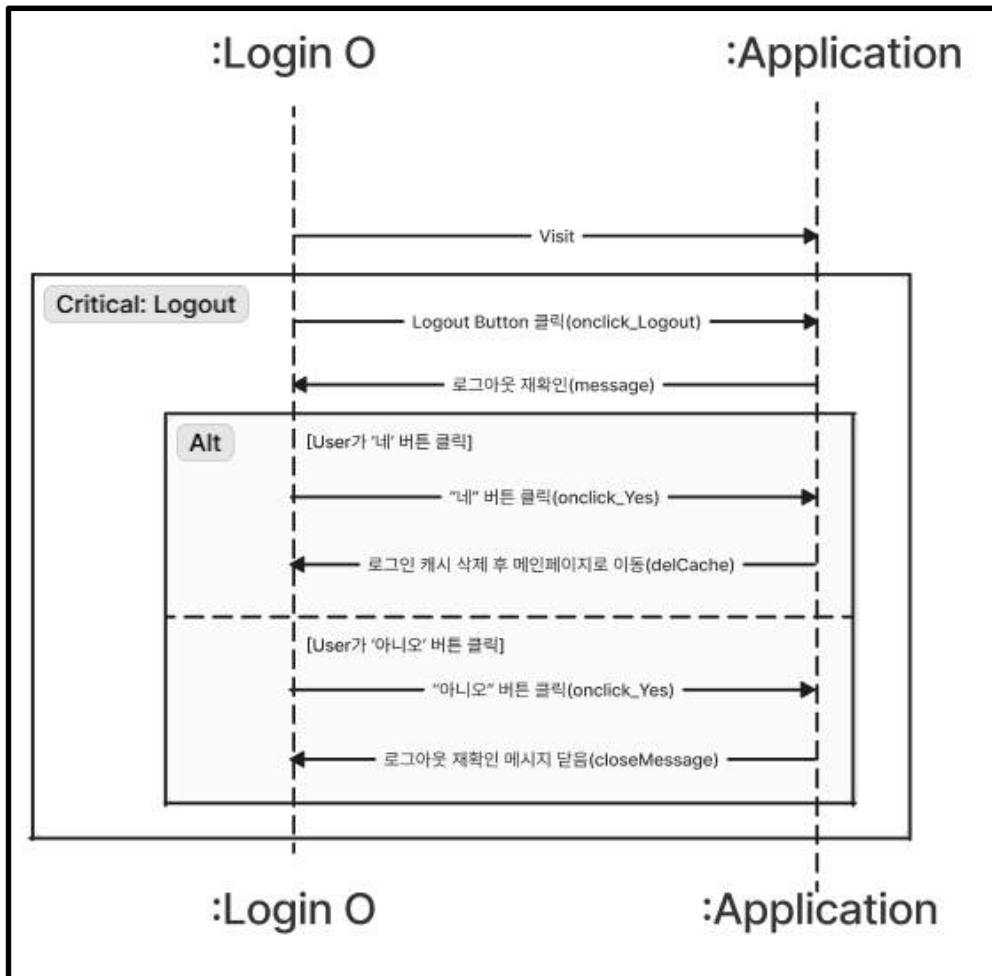
<Figure2. domain model>

해당 모델은 “이슈 관리 시스템”의 도메인 모델이다. 시스템의 주요 목적은 프로젝트 내에 있는 이슈들을 관리하는 데에 있다. 시스템은 프로젝트 리더(PL), 개발자(Developer), 테스터(Tester)와 같은 다양한 사용자 역할을 지원하고, 각 사용자는 시스템 내에서 자신의 계정을 통해 작업을 수행한다.

### <각 Entity간의 관계>

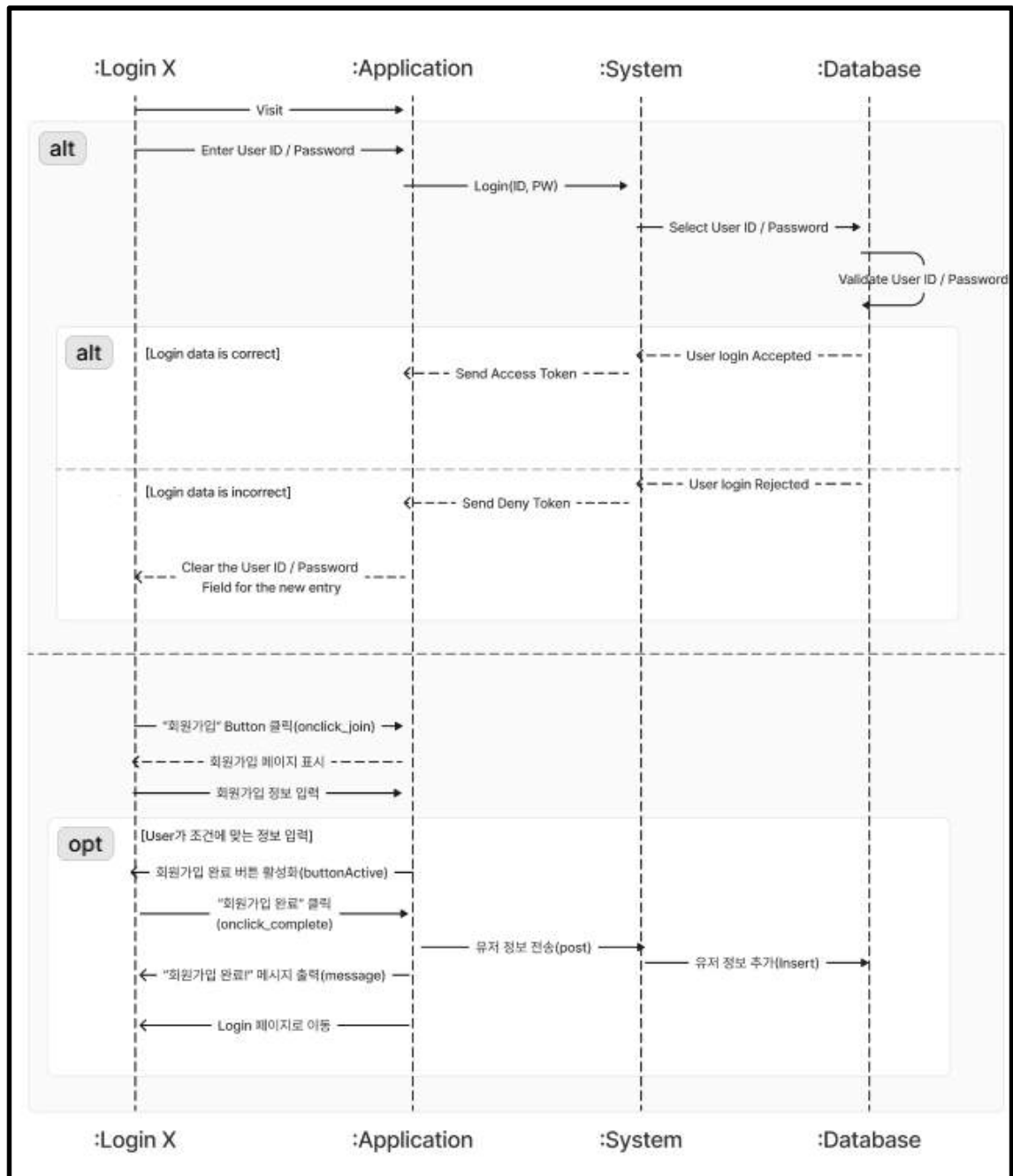
- User-PL, User-Developer, User-Tester: PL, Developer, Tester는 User를 상속 받는다.
- User-Account: User가 없으면 Account도 없는 Composition 관계를 가진다. Account의 속성값으로 ID, PW, nickname이 있다.
- PL-Project: PL은 Project를 관리한다. PL 한 명에게 최소 1개 이상의 Project가 있고, Project 1개당 PL은 한 명이다.
- PL-Tester: PL 한명에 Tester는 0명 또는 무수히 많을 수 있고, Tester 한 명에 PL은 최소 한 명 또는 그 이상이다.
- Developer-Project: 두 엔티티는 Aggregation 관계를 가진다.
- Developer-Issue: 개발자 한 명에 0명 또는 그 이상의 Tester가 있고, Tester 한 명에 0명 또는 1명의 개발자가 있다.
- Project-Issue: 프로젝트가 없으면 이슈가 없는 Composition 관계를 갖는다. 프로젝트 한 개에 이슈가 없거나 무수히 많을 수 있으며 이슈 한 개에 프로젝트는 한 개다.
- Issue-Comment: 이슈가 없으면 코멘트가 없는 Composition 관계를 갖는다. 프로젝트 한 개에 이슈가 없거나 무수히 많을 수 있으며 이슈 한 개에 프로젝트는 한 개다.

## 2.4. SSD(System Sequence Diagram)



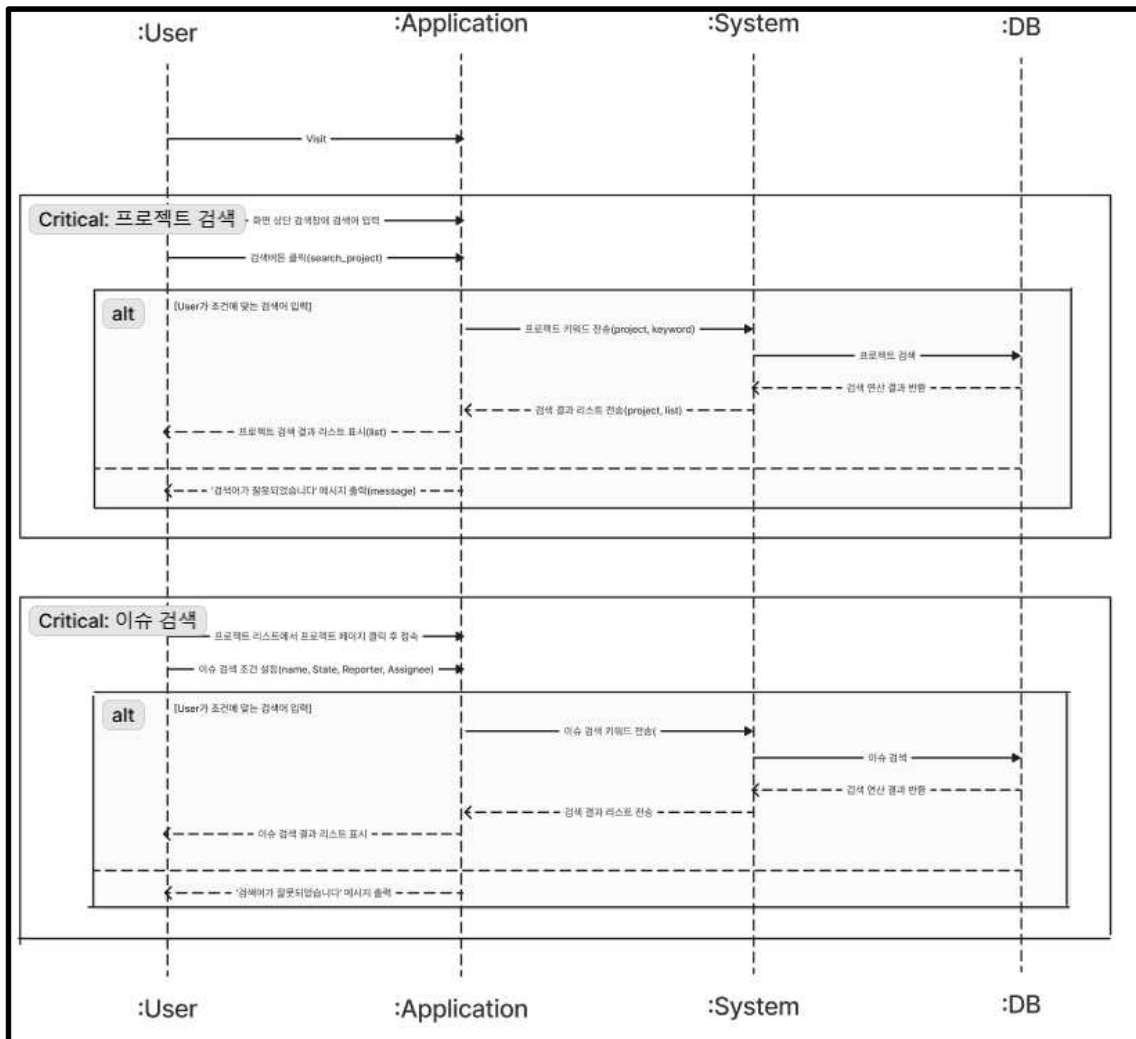
<Figure3. SSD-LoginO>

<Figure3>은 사용자(로그인 O)와 Appliaction간의 상호 작용을 순서대로 나타낸 로그인 아웃 과정을 나타내는 SSD이다. 이 과정은 중간에 다른 메시지가 들어올 수 없도록 Critical Section으로 묶었으며, 다시 Alt Section을 통해 조건에 따라 메시지가 수행된다.



<Figure4. SSD-LoginX>

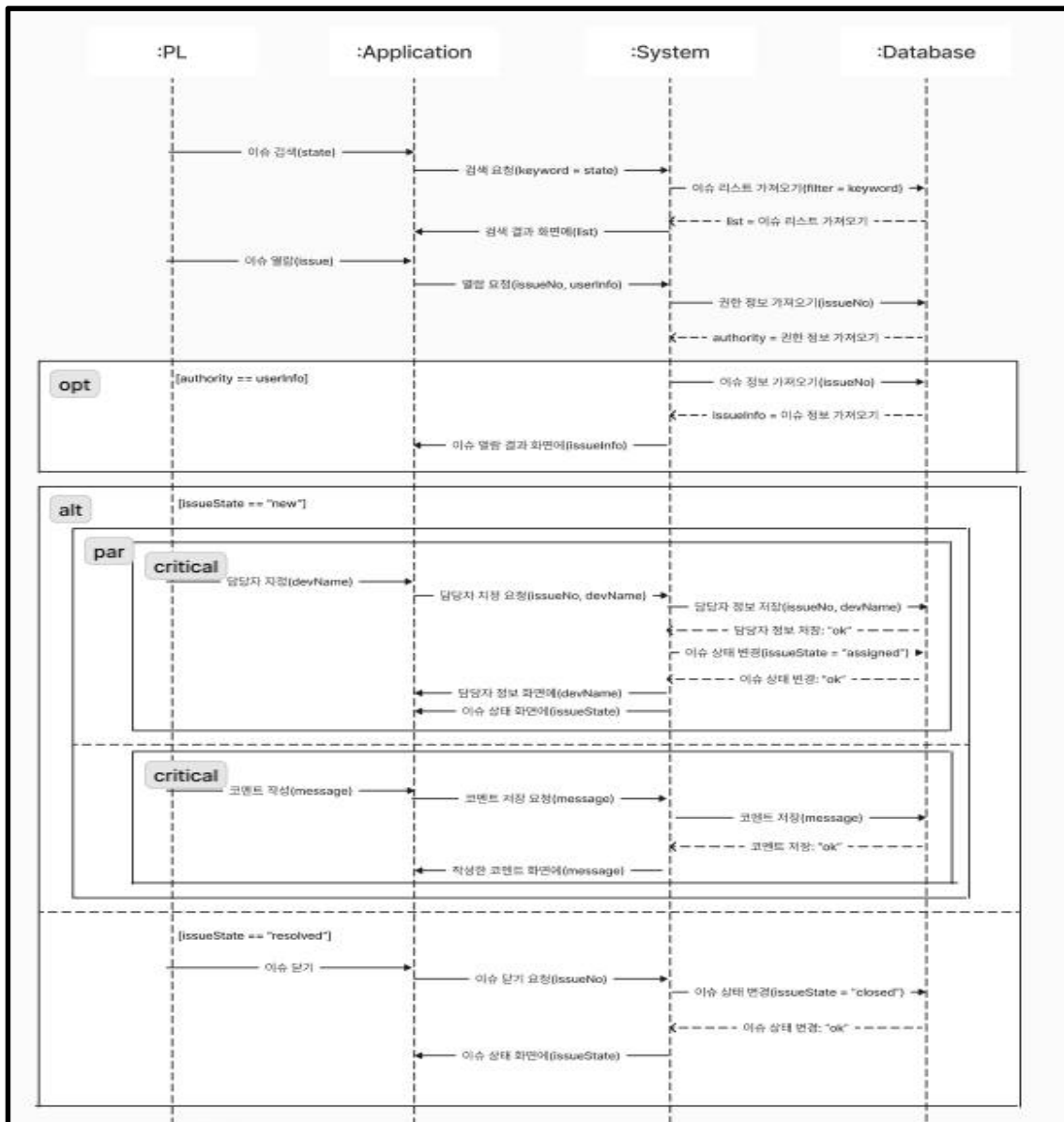
<Figure4>는 사용자(로그인 X), Application, System, Database간의 상호 작용을 나타낸다. 이 다이어그램은 로그인인 되어 있지 않은 사용자가 로그인 및 회원가입을 하는 과정으로 주어진 조건에 맞게 메시지가 수행된다.



<Figure5. SSD-User>

<Figure5>는 사용자, Application, System, Database간의 상호 작용을 나타낸 다이어그램이다. Critical Section을 통해 프로젝트 검색과 이슈 검색으로 각각의 과정에 따라 Section이 나누어졌다. Critical:프로젝트 검색 Section에서는 사용자가 Application 화면 상단에 있는 검색창에 검색어를 입력하고 검색 버튼을 클릭하면 Application에서는 System으로 프로젝트 키워드를 전송한다. System은 다시 Database에서 프로젝트를 검색하게 되고, 검색 결과를 반환한다. 이때 사용자가 조건에 맞는 검색어를 입력했는지에 따라 Alt Section으로 나뉘어 메시지가 다르게 수행된다.

Critical:이슈 검색 Section에서도 앞선 Section과 마찬가지로 사용자가 조건에 맞는 검색어를 입력했는지에 따라 alt Section으로 나뉘고 각 조건에 따라 메시지가 수행된다.



<Figure6. SSD-PL>

<Figure6>은 PL(Project Leader), Application, System, Database간의 상호 작용을 나타낸 다이어그램이다.

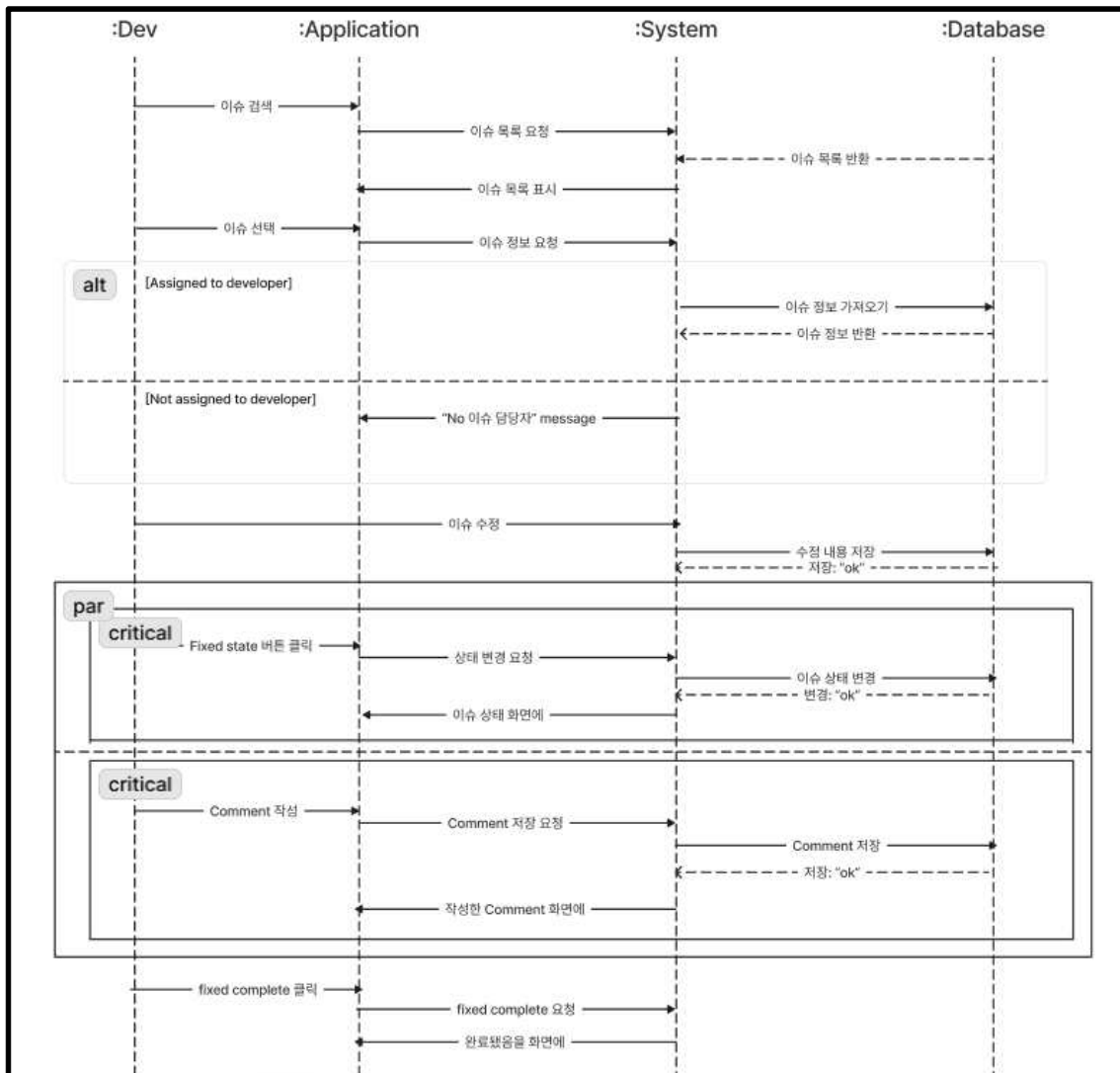
- 이슈 검색(state)
- PL이 이슈의 상태를 조회한다.
- Application이 이를 받아 검색 요청(keyword와 state)을 System에 보낸다.
- System은 데이터베이스에서 해당 조건에 맞는 이슈 리스트를 검색(filter는 keyword)한다.
- 검색된 이슈 리스트가 System에서 Application으로 전달되고, Application은 이를 PL로 반환한다.

- 이슈 열람(issue)
  - PL이 특정 이슈를 열람한다.
  - Application이 해당 이슈 정보를 조회하기 위해 System에 요청(issueNo, userInfo)을 보낸다.
  - System은 데이터베이스에서 이슈 정보를 가져와 이를 Application에 전달한다.
  - Application은 이를 PL로 반환한다.
  
- 이슈 할당(issueState == "new").
  - Application은 이슈 상태가 "new"인지 확인한 후, 담당자 지정 요청(issueNo, devName)을 System에 보낸다.
  - System은 데이터베이스에 담당자 정보를 저장하고, 이슈 상태를 "assigned"로 변경한다.
  - 이 과정이 정상적으로 완료되면 "ok" 응답이 Application을 통해 PL로 전달된다.
  
- 코멘트 작성(message)
  - PL이 이슈에 대한 코멘트를 작성한다.
  - Application이 코멘트 저장 요청(issueNo, message)을 System에 보낸다.
  - System은 데이터베이스에 코멘트를 저장하고 "ok" 응답을 Application에 전달한다.
  - Application은 이를 PL로 반환한다.
  
- 이슈 종료(issueState == "resolved")
  - PL이 이슈를 해결한다.
  - Application이 이슈 상태를 "resolved"로 변경 요청(issueNo)을 System에 보낸다.
  - System은 데이터베이스에서 이슈 상태를 "closed"로 업데이트한다.
  - 이 과정이 정상적으로 완료되면 "ok" 응답이 Application을 통해 PL로 전달된다.
  
- opt (옵션)
  - authority가 userInfo와 동일한지 확인한다. 동일할 경우에만 이슈 열람을 진행한다.
- alt (대안 흐름)
  - 이슈 상태가 "new"일 경우 새로운 이슈 할당을 진행한다.
- par (병렬 흐름)
  - 이슈 할당과 코멘트 작성을 병렬로 처리한다.
- critical (중요 단계)
  - 이슈 할당과 코멘트 작성의 세부 단계는 중요한 단계로 표시된다.

#### □ 요약

이 시퀀스 다이어그램은 시스템 내에서 이슈를 검색, 열람, 상태 변경, 코멘트 추가 및 종료하는 전체 프로세스를 보여준다. 각 단계는 명확한 메시지 교환을 통해 이루어지며, 다양한 조건과 흐름 제어를 통해 유연하게 처리된다.





<Figure7. SSD-Dev>

- 이슈 검색
  - Developer가 이슈를 조회한다.
  - Application이 이를 받아 이슈 목록 요청을 System에 보낸다.
  - System은 데이터베이스에서 해당 조건에 맞는 이슈 목록을 반환한다.
  - 반환된 이슈 목록이 System에서 Application으로 전달되고, Application은 이를 Developer로 반환한다.
- 이슈 열람
  - Developer가 특정 이슈를 선택한다.
  - Application이 해당 이슈 정보를 조회하기 위해 System에 이슈 정보 요청을 보낸다.
  - System은 데이터베이스에서 이슈 정보를 가져와 이를 Application에 전달한다.
  - Application은 이를 Developer로 반환한다.

- 이슈 수정

- Developer가 이슈 수정을 하면 System에 바로 반영된다.

- System은 수정된 내용을 데이터베이스에 저장한다.

- 이슈 상태 변경

- Developer가 "Fixed state"버튼을 클릭하면 Application이 이를 받아 변경 요청을 System에 보낸다.

- System은 데이터베이스에 상태 정보를 저장하고 Application에 변경된 상태를 반환한다.

- 코멘트 작성

- Developer가 이슈에 대한 코멘트를 작성한다.

- Application이 코멘트 저장 요청을 System에 보낸다.

- System은 데이터베이스에 코멘트를 저장하고, 작성한 코멘트를 Application에 전달한다.

alt (대안 흐름)

- 할당된 개발자인 경우 이슈 정보를 가져온다.

par (병렬 흐름)

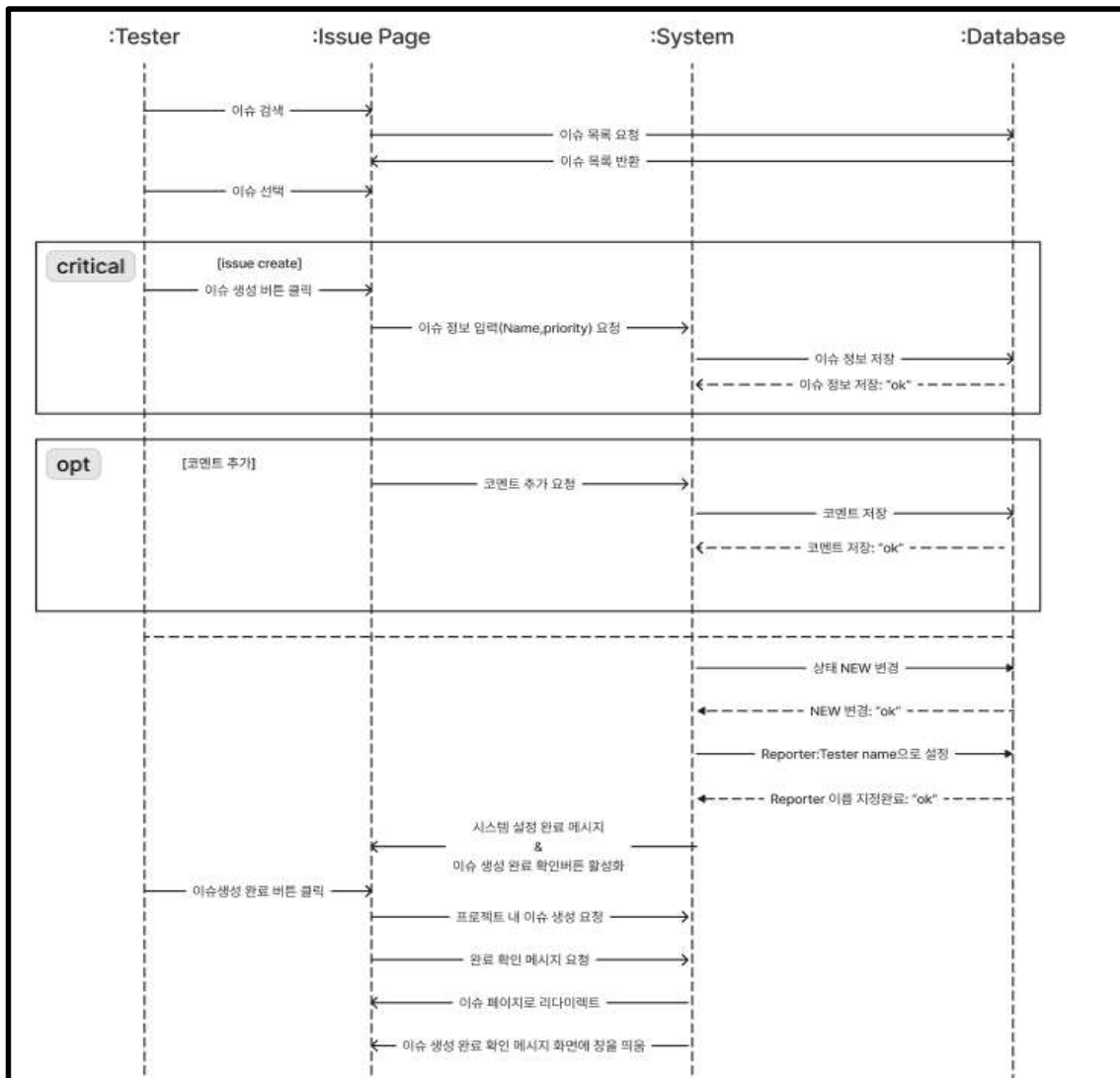
- 이슈 상태 변경과, 코멘트 작성을 병렬로 처리한다.

critical (중요 단계)

- 이슈 상태 변경과, 코멘트 작성의 세부 단계는 중요한 단계로 표시된다.

#### □ 요약

이 시퀀스 다이어그램은 시스템 내에서 이슈를 검색, 선택, 상태 변경, 조건에 따른 진행, 코멘트를 추가하는 전체 프로세스를 보여준다. 각 단계는 명확한 메시지 교환을 통해 이루어지며, 다양한 조건과 흐름 제어를 통해 유연하게 처리된다.



<Figure8. SSD-Tester>

- 이슈 검색

-Issue Page에서 이를 받아 이슈 목록 요청을 System을 거쳐 데이터베이스로 보낸다.

-데이터베이스에서 이슈 목록을 반환한다.

- 이슈 생성 및 코멘트 작성

-Tester가 이슈 생성을 한다.

-Issue Page에서 이슈 정보를 입력하고 System은 데이터베이스로 이슈 정보 저장을 보낸다.

-Issue Page에서 코멘트 추가 요청이 들어오면 System은 데이터베이스에 코멘트를 저장하고 "ok" 메시지를 보낸다.

- 이슈 상태 변경

-System이 이슈의 상태를 New로 변경하고 데이터베이스에 저장하고, Reporter는 데이터베이스에 tester 이름으로 설정된다.

-System은 Issue Page에 이슈 생성 완료 확인 버튼을 반환하고, 확인 메시지들을 주고 받는다.

## □ 요약

이 시퀀스 다이어그램은 시스템 내에서 이슈를 검색, 선택, 상태 변경, 생성, 코멘트 추가등의 전체 프로세스를 보여준다. 각 단계는 명확한 메시지 교환을 통해 이루어지며, 다양한 조건과 흐름 제어를 통해 유연하게 처리된다.

## ※Operation Contract

많은 기능들이 있기에 기능별 Operation contract가 있지만 대표적으로 이슈 생성과 이슈 상태 변화(=assigned)를 예시로 설명할 수 있다.

Operation:	이슈 생성
Cross References:	Issue, User(Tester), Project
Preconditions:	사용자 객체는 시스템에 로그인된 상태여야 한다. 사용자는 이슈 생성 권한이 있어야 한다. 프로젝트가 존재해야 한다.
Postconditions:	새로운 이슈 객체가 생성된다. 새로 만들어진 이슈는 프로젝트의 이슈 목록에 추가된다, 이슈의 상태는 "New"로 설정된다.

→ 이 작업은 사용자(Tester)가 시스템에 로그인하고 있으며, 해당 사용자가 이슈를 생성할 수 있는 권한을 가지고 있어야 하며, 또한 이슈가 생성될 프로젝트가 존재해야 하는 것을 전제 조건으로 한다. 이러한 조건들이 만족될 때, 새로운 이슈 객체가 생성되고, 해당 프로젝트의 이슈 목록에 추가된다. 이때 생성된 이슈의 상태는 'New'로 설정된다.

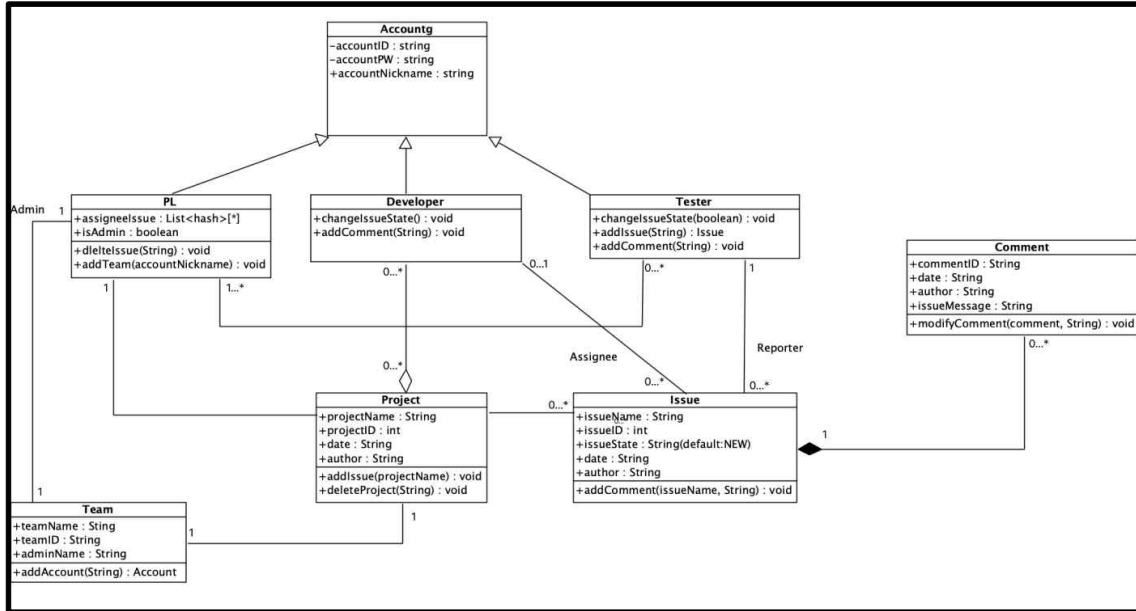
Operation:	State를 assigned로 변경
Cross References:	Issue, User(PL)
Preconditions:	PL이 원하는 이슈에 대한 이슈 열람 화면에 접속되어 있어야 한다. System은 권한이 있는 PL에게 "Assigned State" 버튼을 표시해 준다.
Postconditions:	"이슈의 상태가 Assigned로 변경되었습니다!" 안내문구 띄워준다. 화면에서 이슈 상태가 Assigned로 보여진다.

→ 이 기능은 PL이 특정 이슈의 상태를 "Assigned"로 변경할 수 있도록 하는 작업이다. PL이 원하는 이슈에 대한 열람 화면에 접근하면, 시스템은 해당 PL에게만 이슈 상태를 "Assigned"로 변경할 수 있는 버튼을 표시한다. 이 버튼을 누르면 시스템은 이슈의 상태를 "Assigned"로 변경하고, 성공적으로 변경되었음을 알리는 안내 문구를 표시한다. 또한, 이슈 열람 화면에서 해당 이슈의 상태가 "Assigned"로 업데이트되어 표시된다.

### 3. 설계

#### 3.1. Diagram

##### 3.1.1. Class Diagram



<Figure9. Class Diagram>

Class Diagram은 객체 지향 모델링에서 자주 사용되는 유형의 정적 구조 다이어그램이다. 시스템의 클래스 구조와 그 클래스들 사이의 관계를 시각화한 것으로 UML의 일부로, 소프트웨어 설계 과정에서 클래스의 구조와 시스템의 객체 간 상호 작용을 표현하는 데 사용된다.

Class	Attribute	설명
Account	-accountID:string	계정의 아이디를 나타내는 private속성이고, String 타입이다.
	-accountPW:string	계정의 비밀번호를 나타내는 private속성이고, String 타입이다.
	+accountNickname:string	계정의 닉네임을 나타내는 public속성이고, String 타입이다.
PL	+assigneeIssue:List<hash>[*]	public 속성으로, 해시 값을 가진 객체의 리스트를 나타낸다. 리스트는 0개 이상의 요소를 포함할 수 있다.
	+isAdmin:boolean	public 속성으로, 사용자가 관리자 권한을 가지고 있는지 여부를 나타내는 논리타입이다. true는 관리자, false는 비관리자를 의미한다.

Project	+projectName:String	프로젝트 이름을 나타내고, String 타입이며 Public속성이다.
	+projectID:int	프로젝트 아이디를 나타내고, int 타입이며 Public속성이다.
	+date:String	프로젝트의 날짜를 나타내고, String 타입이며 Public속성이다.
	+author:String	프로젝트의 작성자를 나타내고, String 타입이며 Public속성이다.
Team	+teamName:String	Team의 이름을 나타내며, String 타입으로 저장된다.
	+teamID:String	Team의 아이디를 나타내며, String 타입으로 저장된다.
	+adminName:String	Team의 관리자를 나타내며, String 타입으로 저장된다.
Issue	+issueName:String	이슈의 이름을 나타내며, String 타입으로 저장된다.
	+issueID:int	이슈의 아이디를 나타내며, int 타입으로 저장된다.
	+issueState:String(default:NEW)	이슈의 상태를 나타내며, String 타입으로 저장된다. 기본값은 "NEW"이다. 이슈의 현재 상태(예: new, closed, resolved등)를 나타내는 데 사용됩니다.
	+date:String	속성은 이슈가 생성되거나 업데이트된 날짜를 나타내며, String 타입으로 저장된다.
	+author:String	속성은 이슈를 생성한 사람의 이름을 나타내며, String 타입으로 저장된다.
Comment	+commentID:string	comment의 아이디를 나타내며, string 타입으로 저장됩니다. 댓글을 구별하기 위해 사용됩니다.
	+date:String	comment가 작성되거나 업데이트된 날짜를 나타내며, String 타입으로 저장된다.
	+author:String	comment를 작성한 사람의 이름을 나타내며, String 타입으로 저장된다.
	+issueMessage:String	comment의 내용을 나타내며, String 타입으로 저장된다. comment에서 전달하고자 하는 메시지나 정보를 담기 위해 사용됩니다.

<Attribute>

Class	Method	설명
PL	+deleteIssue(String):void	이슈를 삭제하는 기능을 수행한다. 이슈 아이디 또는 이름과 같은 정보를 String 형태로 매개변수로 받아, 해당 이슈를 삭제한다.
	+addTeam(accountNickname):void	팀 멤버의 닉네임을 String 형태로 매개변수로 받아, 해당 팀 멤버를 팀에 추가한다.
Developer	+changeIssueState():void	이슈 상태를 변경하는 기능을 수행한다. 이슈 상태를 변경하기 위해 이슈 ID와 새로운 상태 값을 매개변수로 받을 수 있다.
	+addComment(String):void	Comment를 추가하는 기능을 수행한다. String 타입의 매개변수를 받는다.
Tester	+changeIssueState(boolean):void	Issue의 상태를 변경하는 기능을 수행한다. boolean 타입의 매개변수를 받아, Issue의 상태를 변경한다.
	+addIssue(string):Issue	Issue를 추가하는 기능을 수행한다. string 타입의 매개변수를 받아, Issue를 생성하고 이를 반환한다.
	+addCommnet(string):void	Commnet을 추가하는 기능을 수행한다. string 타입의 매개변수를 받아, 해당 내용을 Commnet로 추가한다.
Project	+addIssue(projectName):void	Issue를 추가하는 기능을 수행한다. projectName이라는 매개변수를 받아서 해당 프로젝트에 Issue를 추가한다.
	+deleteProject(String):void	프로젝트를 삭제하는 기능을 수행한다. 삭제할 프로젝트의 이름 또는 식별자를 나타내는 문자열 매개변수를 받는다.
Team	+addAccount(String): Account	계정을 추가하는 기능을 수행한다. String 타입의 매개변수를 받아, 이를 사용하여 새로운 Account 객체를 생성하고 반환한다.
Issue	+addComment(issueName,String):void	issue에 대한 comment를 추가하는 기능을 수행한다. 첫 번째 매개변수로 issueName을 받아 해당 issue를 식별하고, 두 번째 String 타입의 매개변수로 comment의 내용을 받는다.

<Method>

### 3.1.OOAD/GRASP 패턴/설계 원칙

OOAD/GRASP 패턴/설계 원칙의 적용은 Class Diagram을 통해 설명할 수 있다.

#### <객체 지향 설계 원칙 적용/OOAD>

##### 1. 단일 책임 원칙(SRP)

- 적용 부분: Account, Project, Issue, Comment class
- 설명: 각 클래스는 하나의 책임만 가지도록 설계되었다. Account 클래스는 계정 관리, Project 클래스는 프로젝트 관리, Issue 클래스는 이슈 관리, Comment 클래스는 댓글 관리에 대한 책임을 가진다.
- 이유: 각 클래스가 하나의 책임만 가지도록 하여 변경이 필요할 때 영향 범위를 최소화하고, 유지보수를 용이하게 한다.

##### 2. 개방=폐쇄 원칙(OCF)

- 적용 부분: Developer, Tester, PL class
- 설명: 이 클래스들은 Account 클래스를 상속받아 확장되며, 새로운 역할이 추가될 때 기존 코드를 수정하지 않고 확장할 수 있도록 설계되었다.
- 이유: 기존 코드를 수정하지 않고 기능을 확장할 수 있어 시스템의 안정성을 높이고, 새로운 요구사항에 유연하게 대응할 수 있다.

##### 3. 리스코프 치환 원칙(LSP)

- 적용 부분: PL, Developer, Tester class
- 설명: 이 클래스들은 모두 Account 클래스를 상속받아 Account 타입의 객체로 대체 가능하며, 올바르게 기능을 수행한다.
- 이유: 다형성을 올바르게 활용하여 코드의 재사용성을 높이고, 상속 구조의 일관성을 유지 한다.

##### 4. 인터페이스 분리 원칙(ISP)

- 적용 부분: Account 클래스와 PL, Developer, Tester class
- 설명: 각 역할별로 필요한 기능만을 정의하고, 불필요한 메서드를 포함하지 않도록 분리하여 설계되었다.
- 이유: 클라이언트가 자신이 사용하지 않는 메서드에 의존하지 않도록 하여 인터페이스의 응집도를 높이고, 구현 클래스의 복잡도를 낮춘다.

##### 5. 의존성 역전 원칙(DIP)

- 적용 부분: Project, Issue class
- 설명: Project 클래스와 Issue 클래스가 서로 직접 의존하지 않고, 각각의 클래스 내부에서 의존 관계를 설정하도록 설계되었다.
- 이유: 고수준 모듈이 저수준 모듈에 의존하지 않고, 추상화된 인터페이스에 의존하도록 하여 시스템의 유연성과 테스트 용이성을 높인다.



## <GRASP>

### 1. 정보 전문가(Information Expert)

→ 적용 부분: Project, Issue, Comment class

→ 설명: 정보 전문가 패턴은 필요한 정보를 가장 잘 알고 있는 객체에 책임을 할당하는 것이다. Project 클래스는 프로젝트에 대한 정보를, Issue 클래스는 이슈에 대한 정보를, Comment 클래스는 댓글에 대한 정보를 가지고 있다.

→ 이유: 책임을 가장 잘 수행할 수 있는 전문가에게 할당하여 시스템의 응집도를 높이고, 효율성을 증대한다.

### 2. 창조자(Creator)

→ 적용 부분: PL, Developer, Tester class

→ 설명: 창조자 패턴은 특정 객체를 생성할 책임을 가장 적절한 클래스에 할당하는 것이다. PL, Developer, Tester 클래스는 각각의 역할에 따라 이슈를 생성, 수정, 상태 변경 등의 관리를 하는 책임을 가진다.

→ 이유: 객체 생성 책임을 명확히 하여 코드의 가독성을 높이고, 객체 생성 로직을 한 곳에 모아 유지보수를 용이하게 한다.

### 3. 제어자(Controller)

→ 적용 부분: PL class

→ 설명: 제어자 패턴은 시스템 사건을 처리할 책임을 한 클래스에 집중시키는 것이다.

PL 클래스는 팀 관리와 이슈 할당의 주요 사건을 제어한다.

→ 이유: 시스템의 주요 제어 로직을 한 곳에 집중시켜 코드의 응집도를 높이고, 이벤트 처리 로직을 명확히 한다.

### 4. 저수준 정책(Low Coupling)

→ 적용 부분: Account와 관련된 모든 하위 class

→ 설명: 저수준 정책 패턴은 클래스 간의 결합도를 낮추는 것이다. PL, Developer, Tester 클래스는 모두 Account 클래스를 상속받아 기본적인 계정 정보를 공유하지만, 서로 독립적으로 동작한다.

→ 이유: 클래스 간의 결합도를 낮추어 각 클래스의 독립성을 유지하고, 시스템의 유연성과 재사용성을 높인다.

### 5. 고응집도(High Cohesion)

→ 적용 부분: Issue, Comment class

→ 설명: 고응집도 패턴은 관련 기능들을 한 클래스에 모아 응집도를 높이는 것이다. Issue 클래스는 이슈 생성, 상태 변경, 댓글 추가 등 이슈와 관련된 모든 기능을, Comment 클래스는 댓글 생성, 수정 등 댓글과 관련된 모든 기능을 담당한다.

→ 이유: 관련 기능들을 한 클래스에 모아 응집도를 높이고, 클래스의 역할을 명확히 하여 유지보수를 용이하게 한다.

## 6. 가벼운 책임(Low Coupling)

- 적용 부분: Project와 Issue class의 관계
- 설명: 가벼운 책임 패턴은 클래스 간의 결합도를 낮추는 것이다. Project 클래스는 여러 Issue 객체와 연관되지만, Issue 객체는 프로젝트 내에서 독립적으로 존재할 수 있다.
- 이유: 클래스 간의 결합도를 낮추어 각 클래스의 독립성을 유지하고, 시스템의 유연성과 재사용성을 높인다.

## 7. 변형자(Polymorphism)

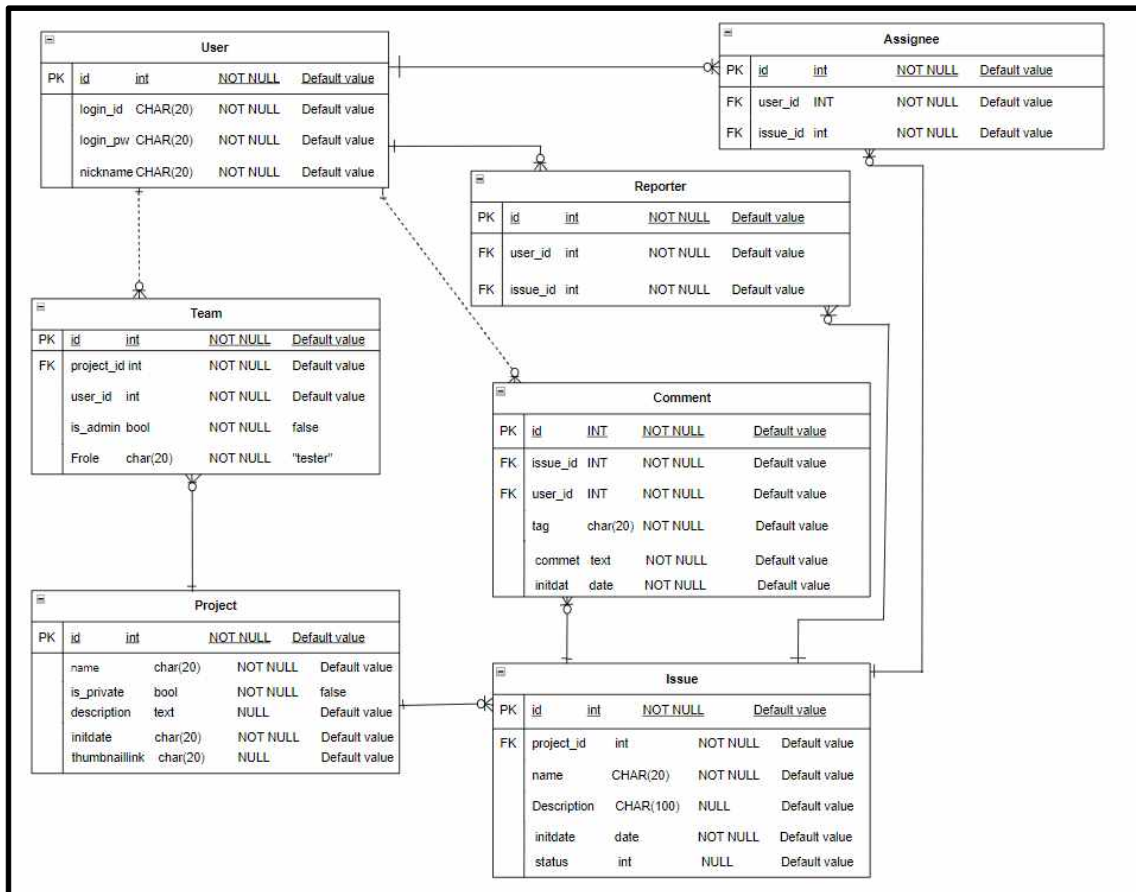
- 적용 부분: Account 클래스를 상속받는 PL, Developer, Tester class
- 설명: 변형자 패턴은 다형성을 통해 코드의 재사용성을 높이는 것이다. PL, Developer, Tester 클래스는 모두 Account 클래스를 상속받아 다형성을 통해 각기 다른 역할을 수행한다.
- 이유: 다형성을 활용하여 코드의 재사용성을 높이고, 확장성을 제공한다.

## 8. 보호 변형자(Protected Variations)

- 적용 부분: addComment 메서드
- 설명: 보호 변형자 패턴은 변화를 캡슐화하여 외부에 미치는 영향을 최소화하는 것이다. addComment 메서드는 Issue와 Comment 클래스에서 사용되며, 각기 다른 방식으로 댓글을 추가할 수 있다. 이 메서드는 인터페이스를 통해 다양한 구현을 허용하면서도 외부로부터의 변화를 보호한다.
- 이유: 변화를 캡슐화하여 외부에 미치는 영향을 최소화하고, 시스템의 안정성을 높인다.

### 3.3. 추가적인 UML Diagram

#### 3.3.1.ERD(Entity Relationship Diagram)



<Figure10. Entity Relationship Diagram>

ERD는 데이터베이스의 구조를 시각적으로 표현하는 데 사용되는 다이어그램이다. 이 다이어그램은 데이터베이스 설계 과정에서 중요한 역할을 하며, 개체(Entity), 속성(Attribute), 관계(Relationship) 등의 핵심 요소를 통해 데이터 모델을 구성하고 표현한다.

위 그림에서 개체는 데이터베이스에 저장되는 정보의 단위로 유저, 팀, 프로젝트, 리포터, 코멘트, 이슈, 할당자가 있으며, 각 개체마다 PK(primary key), FK(foreign key)를 갖고 있다. ERD에서 entity들은 Cardinality를 가지게 되는데 위 그림에서의 각 entity들은 모두 zero or one or many의 관계를 가지고 있다.

다음은 각 entity간의 PK, FK 참조 관계이다.

- 팀 entity는 유저 entity의 PK를 참조하고, 프로젝트 entity의 PK도 참조한다.
- 이슈 entity는 프로젝트 entity의 PK를 참조한다.
- 코멘트 entity와 리포터 entity 그리고 할당자 entity는 유저 entity의 PK를 참조한다.
- 코멘트 entity, 리포터 entity, 할당자 entity 모두 이슈 entity의 PK를 참조한다.

## 4. 구현

### 4.1. 주요 기능 Screen shot

```
package com.issuestation.Service.IssueService;

import ...

@Service
public class IssueInfoServiceImpl {

    private final IssueRepository issueRepository;
    private final AssigneeRepository assigneeRepository;
    private final ReporterRepository reporterRepository;
    private final UserRepository userRepository;

    public IssueInfoServiceImpl(IssueRepository issueRepository, AssigneeRepository assigneeRepository, ReporterRepository reporterRepository, UserRepository userRepository) {
        this.issueRepository = issueRepository;
        this.assigneeRepository = assigneeRepository;
        this.reporterRepository = reporterRepository;
        this.userRepository = userRepository;
    }

    @Transactional(readOnly = true)
    public IssueResponseDto.JoinIssueInfoResponseDto getIssueDetails(Long issueId) {
        Issue issue = issueRepository.findById(issueId).orElseThrow(() -> new RuntimeException("Issue not found"));

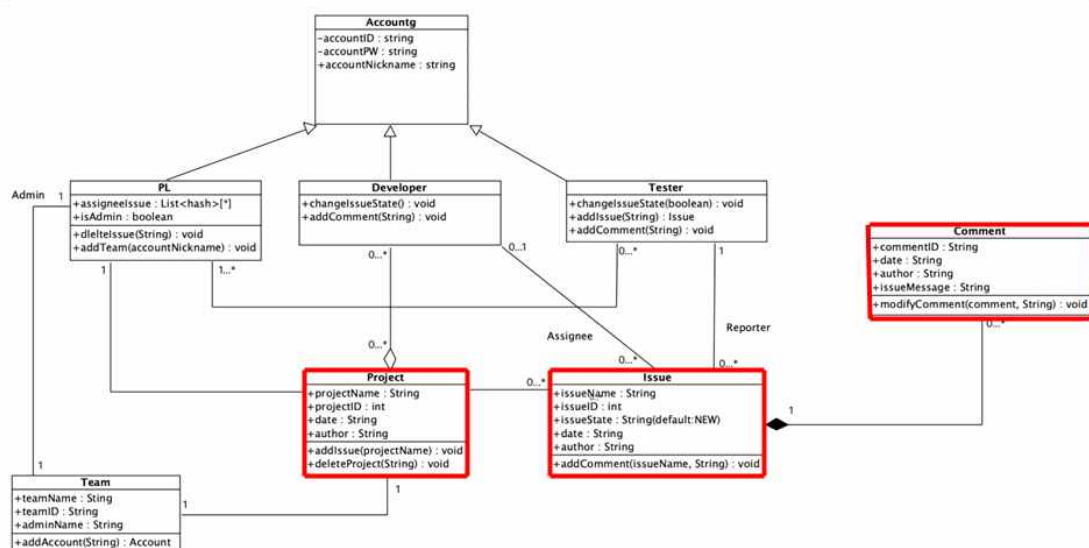
        Assignee assigneeId = assigneeRepository.findDistinctFirstByIssueId(issueId).orElse(null);

        Reporter reporterId = reporterRepository.findDistinctFirstByIssueId(issueId).orElse(null);

        String assigneeNickname = "not assigned";
        String reporterNickname = "not reported";
        if (assigneeId != null) {
            assigneeNickname = userRepository.findById(assigneeId.getUser().getId()).get().getNickname();
        }
        if (reporterId != null) {
            reporterNickname = userRepository.findById(reporterId.getUser().getId()).get().getNickname();
        }

        return new IssueResponseDto.JoinIssueInfoResponseDto(
            issue.getId(),
            issue.getName(),
            issue.getDescription(),
            issue.getStatus(),
            issue.getProject().getId(),
            String.valueOf(issue.getInitdate()),
            String.valueOf(issue.getModdate()),
            assigneeNickname,
            reporterNickname
        );
    }
}
```

<Figure 11. IssueInfoServiceImpl>



- issueInfoServiceImpl 클래스는 특정 이슈의 상세 정보를 조회하고 반환하는 서비스를 제공한다.

※클래스 다이어그램과의 연관성※

- ✓ Issue 클래스는 이슈의 기본 정보를 포함한다.
- ✓ Project 클래스와 연관 관계로, 이슈가 속한 프로젝트 정보를 가진다.
- ✓ Comment 클래스와 연관 관계로, 이슈에 달린 코멘트들을 가진다.

```
package com.issuestation.Service.IssueService;

import ...

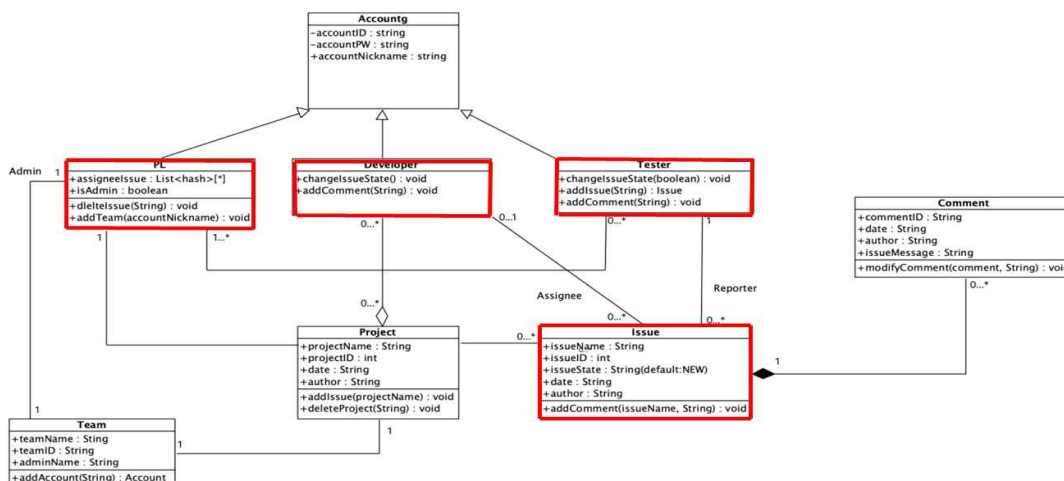
@Service no usages 1 bowook +1
@RequiredArgsConstructor
@Transactional(readOnly = true)
public class IssueStateServiceImpl implements IssueStateService {

    private final IssueRepository issueRepository; 2 usages

    @Override no usages 1 bowook
    @Transactional
    public Issue changeIssueState(IssueRequestDto.JoinIssueStateRequestDto request, long issueId) {
        Issue issue = issueRepository.findById(issueId)
            .orElseThrow(() -> new IllegalArgumentException("Issue not found"));

        Issue updatedIssue = IssueStateConverter.updateIssueState(issue, request.getStatus());
        return issueRepository.save(updatedIssue);
    }
}
```

<Figure12. IssueStateServiceImpl>



- 이 코드는 이슈의 상태를 변경하는 서비스를 구현한 클래스이다.

※클래스 다이어그램과의 연관성※

- ✓ Issue 클래스에서 issueState 필드를 가지고 있다.
- ✓ PL, Developer, Tester 클래스 각각에 changelssueState 메소드가 정의되어 있어, 이슈 상태를 변경할 수 있다.

```
package com.issuestation.Service.IssueService;

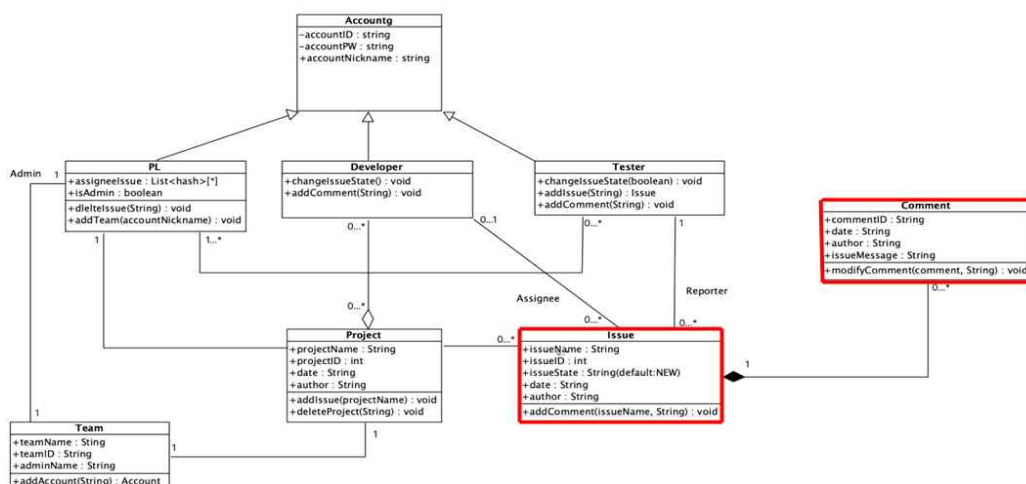
import ...

@Service
@RequiredArgsConstructor
@Transactional(readOnly = true)
public class CommentCreateServiceImpl implements CommentCreateService{

    private final IssueRepository issueRepository;
    private final CommentRepository commentRepository;
    private final TokenProvider tokenProvider;
    private final UserRepository userRepository;

    @Override
    @Transactional
    public Comment createComment(IssueRequestDto.JoinCommentCreateRequestDto request, long issueId, String token) {
        Issue issue = issueRepository.findById(issueId).orElseThrow(() -> new IllegalArgumentException("Invalid Issue ID"));
        long userId = Long.parseLong(tokenProvider.validateJwt(token));
        User userEntity = userRepository.findById(userId).get();
        Comment comment = CommentCreateConverter.toCommentEntity(request, issue, userEntity);
        return commentRepository.save(comment);
    }
}
```

<Figure13. CommentCreateServiceImpl>



- 이 코드는 이슈에 댓글을 생성하는 서비스를 구현한 클래스이다.

※클래스 다이어그램과의 연관성※

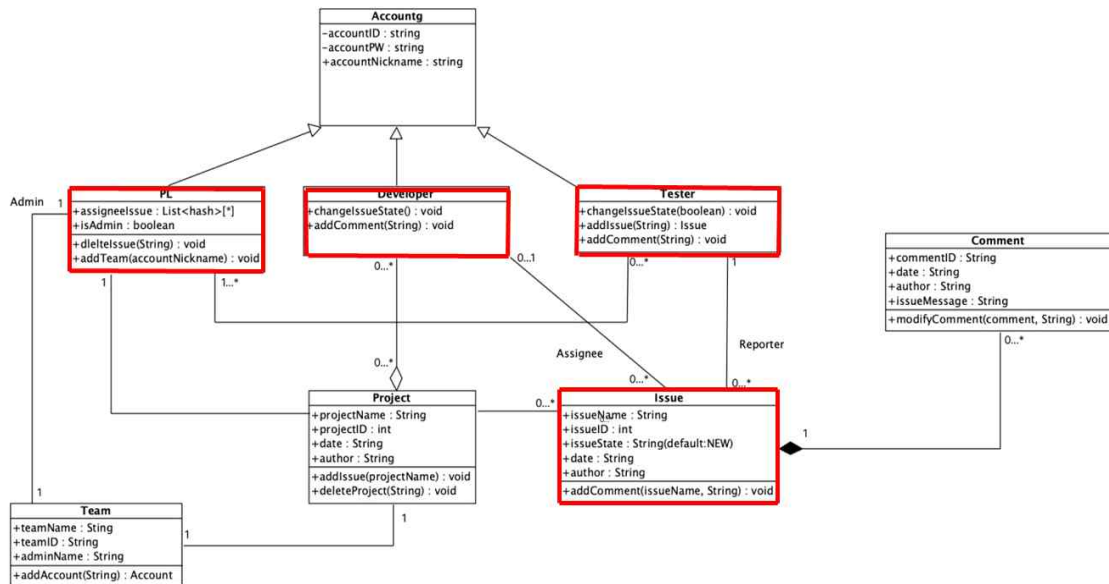
- ✓ Comment 클래스는 코멘트의 정보를 포함한다. commentID, date, author, issueMessage 정보를 가진다.
- ✓ Issue 클래스와의 연관 관계로, 하나의 이슈가 여러 개의 코멘트를 가질 수 있다.

```
public Reporter assignReporter(Long issueId, IssueRequestDto.JoinAssigneeRequestDto assigneeRequest) {
    Optional<Issue> issueOptional = issueRepository.findById(issueId);
    Optional<User> userOptional = userRepository.findByNickname(assigneeRequest.getNickname());

    if (issueOptional.isPresent() && userOptional.isPresent()) {
        Issue issue = issueOptional.get();
        User user = userOptional.get();

        Reporter reporter = Reporter.builder()
            .issue(issue)
            .user(user)
            .build();
        return reportRepository.save(reporter);
    } else {
        throw new RuntimeException("Issue or User not found");
    }
}
```

<Figure14. SetAssigneeService>



- 이 코드는 이슈에 담당자(assignee) 및 보고자(reporter)를 설정하는 서비스를 구현한 클래스이다.

※클래스 다이어그램과의 연관성※

- ✓ Issue 클래스: 담당자 할당 (Assignee는 Developer)
- ✓ PL 클래스: 이슈를 할당(assignIssue)

```
package com.issuestation.Service.IssueService;

import ...

@Service no usages 1 dongheeHan +1
public class IssueSearchService {

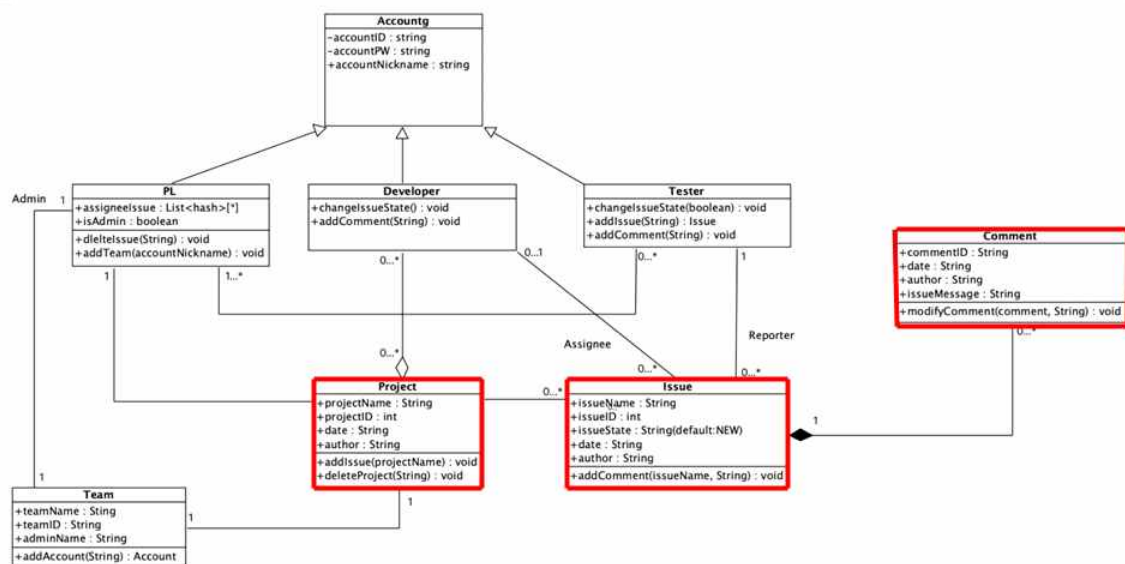
    private static final DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd"); no usages
    @Autowired 3 usages
    private IssueRepository issueRepository;

    public List<IssueResponseDto.JoinIssueSearchResponseDto> searchIssuesByProjectIdNameAndStatus(Long projectId, String name, String status) {
        if (Objects.equals(status, "")) {
            List<Issue> issues = issueRepository.findByProjectIdAndNameContaining(projectId, name);
            return issues.stream().map(this::convertToDto).collect(Collectors.toList());
        } else {
            Status statusEnum = Status.valueOf(status);
            List<Issue> issues = issueRepository.findByProjectIdAndNameContainingAndStatus(projectId, name, statusEnum);
            return issues.stream().map(this::convertToDto).collect(Collectors.toList());
        }
    }

    public List<IssueResponseDto.JoinIssueSearchResponseDto> searchIssuesByProjectId(Long projectId) { no usages 1 dongheeHan
        List<Issue> issues = issueRepository.findByProjectId(projectId);
        return issues.stream().map(this::convertToDto).collect(Collectors.toList());
    }

    private IssueResponseDto.JoinIssueSearchResponseDto convertToDto(Issue issue) { 3 usages 1 dongheeHan
        return IssueResponseDto.JoinIssueSearchResponseDto.builder()
            .id(issue.getId())
            .name(issue.getName())
            .description(issue.getDescription())
            .status(issue.getStatus())
            .projectId(issue.getProject().getId())
            .modDate(String.valueOf(issue.getModdate()))
            .build();
    }
}
```

<Figure15. IssueSearchService>





- 이 코드는 프로젝트 ID, 이슈 이름 및 상태에 따라 이슈를 검색하는 서비스를 구현한 클래스이다.

※클래스 다이어그램과의 연관성※

- ✓ Issue 클래스: 검색 가능한 필드들 (issueName, issueID, issueState)
- ✓ Project, Comment 클래스: 관련 프로젝트 정보와 코멘트 정보 포함 가능

```
package com.issuestation.Service.IssueService;

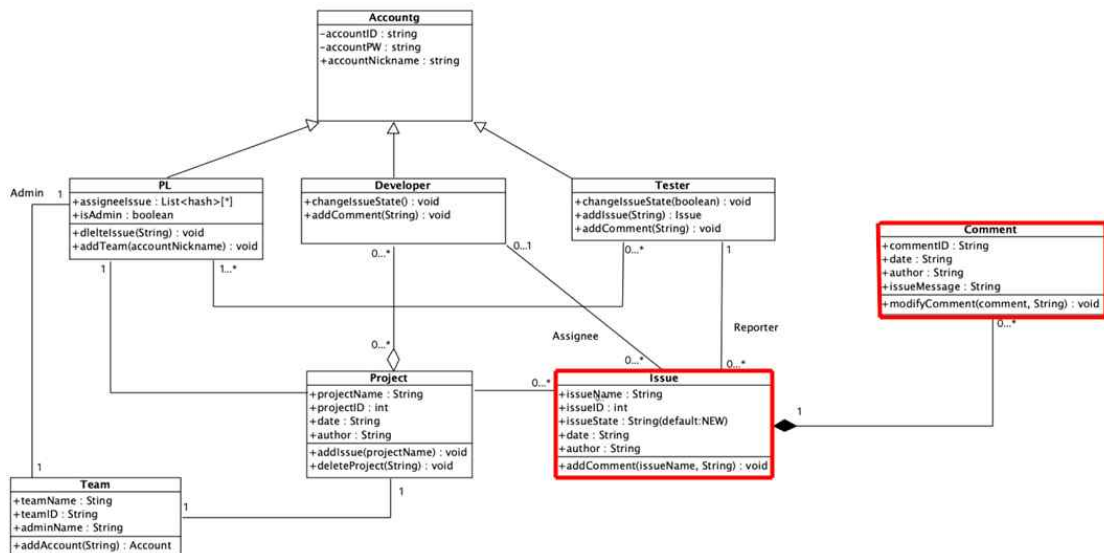
import com.issuestation.Dto.Issue.IssueRequestDto;
import com.issuestation.Dto.Project.ProjectRequestDto;
import com.issuestation.Entity.Issue;
import com.issuestation.Entity.Project;

public interface IssueCreateService { 1 usage  ⤴ dongheeHan +1

    Issue joinIssue(IssueRequestDto.JoinIssueCreateRequestDto request, long projectId); no usages  ⤴ b

}
```

<Figure16. IssueCreateService>



- 이 코드는 이슈를 생성하는 서비스를 구현한 클래스이다.

※클래스 다이어그램과의 연관성※

- ✓ Issue 클래스: 이슈 생성 시 필요한 기본 정보를 포함 (issueName, issueID, issueState, date, author).
- ✓ Project 클래스: 이슈가 특정 프로젝트에 속해 있어야 하므로 Project 클래스와 연관 관계가 있다.

### 4.3 API 기능 명세

기능	Method	Endpoint	Request Body	Request Header	query string	path variable
회원가입	POST	/user/signup	{ nickname : "prom", id "donghoon", pw "qwer1234", }			
닉네임 중복 확인	POST	/user/nickname	{ nickname : "prom" }			
아이디 중복 확인	POST	/user/id	{ id "donghoon" }			
로그인	POST	/user/login	{ id "donghoon", pw "qwer1234" }			
토큰 유저 정보 조회	GET	/user/check		Authorization: accessToken (String)		
이슈 상태 변경	POST	/issue/state/{ id}	{ state "fixed" }	Authorization: accessToken (String)		id: issue id
이슈 생성	POST	/issue/create /{id}	{ name "first issue", description: "haha" }	Authorization: accessToken (String)		id: project id
이슈 상세 정보 조회	GET	/issue/info/{ id}		Authorization: accessToken (String) (선택)		id: issue id
이슈 목록 조회 및 검색	GET	/issue/search {id}		Authorization: accessToken (String) (선택)	name="issue", (선택) sort="desc", (선택) state="fixed" (선택) assignee="pro me", (선택) reporter="pro me" (선택)	
이슈 삭제		/issue/delete /{id}	{ comment: "good", tag: "NEW" }	Authorization: accessToken (String)		id: issue id
코멘트 작성	POST	/issue/comm ent/create/{id }		Authorization: accessToken (String)		id: issue id

프로젝트 생성	POST	/project/create	{ name "test project", description "test", private false }	Authorization: accessToken (String)		
내 프로젝트 목록	GET	/project/my		Authorization: accessToken (String)		
프로젝트 목록 조회 및 검색	GET	/project/search			name="project", (선택) sort="desc", (선택)	
프로젝트 정보 조회	GET	/project/info/{id}		Authorization: accessToken (String) (선택)		id: project id
팀 멤버 추가	POST	/project/team/{id}	{ name "team", isAdmin false }	Authorization: accessToken (String)		id: project id
담당자 할당	POST	/issue/assignee/{id}	{ name "prone" }	Authorization: accessToken (String)	id: issue id	
리포터 할당	POST	/issue/reporter/{id}	{ name "prone" }	Authorization: accessToken (String)	id: issue id	
특정 이슈의 댓글 목록 조회	GET	/issue/comment/{id}			id: issue id	
프로젝트 팀 멤버 조회	GET	/project/team/member/{id}		Authorization: accessToken (String)	id: project id	
프로젝트에서의 역할 조회	GET	/project/role/{id}		Authorization: accessToken (String)	id: project id	
Assignee 알고리즘	GET	/assignee/algorithm/{id}				

☑ 위에 작성된 표는 API 기능 명세 표이다. 새로 추가된 API 기능인 “토큰 유저 정보 조회” 기능은 사용자가 유효한 액세스 토큰을 통해 유저 정보를 조회할 수 있도록 설계되었다.

"토큰 유저 정보 조회" 기능은 사용자가 유저 정보를 안전하게 조회할 수 있는 방법을 제공한다. 이를 통해 애플리케이션 내에서 사용자 인증 및 권한 관리를 보다 효과적으로 수행할 수 있다.

Endpoint: /user/check: 이 엔드포인트는 사용자가 유효한 액세스 토큰을 통해 유저 정보를 조회할 수 있는 경로입니다. 클라이언트는 이 엔드포인트를 호출하여 현재 인증된 사용자의 정보를 요청한다.

Request Header: Authorization: accessToken (String): 클라이언트는 요청 헤더에 'Authorization' 키를 포함시키고, 값으로 유효한 accessToken을 전달한다. 이 토큰은 사용자의 신원을 확인하고 권한을 부여하는 데 사용된다.

## 5. Test Case 수행 내역

### 5.1. Test Case 기본 기능/추가 기능

[AuthControllerTest]

```
@DisplayName("회원 가입 성공") no usages bowook
@Test
void signUpSuccess() throws Exception{
    //given
    SignupDto signupDto = new SignupDto("test", "testN", "1004");
    ResponseDto<String> expectedResponse = ResponseDto.setSuccess("User registered successfully", null);

    given(authService.signUp(signupDto)).willReturn(expectedResponse);

    // when & then
    mockMvc.perform(post("/user/signup")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(signupDto)))
        .andExpect(status().isOk())
        .andExpect(content().string(objectMapper.writeValueAsString(expectedResponse))));
}
```

<signUpSuccess Test>

- 목적: 회원가입이 정상적으로 처리되는지 확인한다. 사용자가 입력한 정보로 회원가입이 완료되었는지 확인한다.
- 테스트 방법: 모의 Authservice를 통해 회원가입 요청을 시뮬레이션하고, 회원가입이 완료되었을 때 예상된 응답이 반환되는지 확인한다.

```
@DisplayName("로그인 성공") no usages bowook
@Test
void loginSuccess() throws Exception {
    // given
    LoginDto loginDto = new LoginDto("testId", "testPw");
    User testUser = User.builder()
        .id(1L)
        .loginId("testId")
        .loginPw("testPw")
        .nickname("testNickname")
        .build();
    String token = "generatedJwtToken";
    LoginResponseDto loginResponseDto = new LoginResponseDto(token, 3600000, testUser);
    ResponseDto<LoginResponseDto> expectedResponse = ResponseDto.setSuccess("Login Success", loginResponseDto);

    given(authService.login(loginDto)).willReturn(expectedResponse);

    // when & then
    mockMvc.perform(post("/user/login")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(loginDto)))
        .andExpect(status().isOk())
        .andExpect(content().string(objectMapper.writeValueAsString(expectedResponse))));
}
```

<loginSuccess Test>

- 목적: 로그인이 정상적으로 되는지 확인한다. 사용자가 입력한 정보로 로그인이 성공하고 토큰이 반환되는지 확인한다.
- 테스트 방법: 모의 Authservice를 통해 로그인 요청을 시뮬레이션하고, 로그인이 완료되었을 때 예상된 응답이 반환되는지 확인한다.

```

@DisplayName("닉네임 중복") no usages bowook
@Test
public void whenNicknameDuplicate_thenReturns400() throws Exception {
    NicknameDto nicknameDto = new NicknameDto("사용중인닉네임");
    NicknameResponseDto nicknameResponseDto = new NicknameResponseDto(true);
    ResponseDto<NicknameResponseDto> responseDto = ResponseDto.setFailed("Nickname already exists", nicknameResponseDto);

    given(authService.nickname(nicknameDto)).willReturn(responseDto);

    mockMvc.perform(post("/user/nickname")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(nicknameDto)))
        .andExpect(status().isOk())
        .andExpect(content().string(objectMapper.writeValueAsString(responseDto)));
}

```

#### <whenNicknameDuplicate\_thenReturns400 Test>

- 목적: 사용자가 입력한 닉네임이 이미 사용 중인지 확인한다. 닉네임이 중복되면 오류 응답이 반환되는지 확인한다.
- 테스트 방법: 모의 AuthService를 통해 닉네임 중복 확인 요청을 시뮬레이션하고, 중복된 경우 오류 응답이 반환되는지 확인한다.

```

@DisplayName("사용 가능한 닉네임") no usages bowook
@Test
public void whenNicknameNotDuplicate_thenReturns200() throws Exception {
    NicknameDto nicknameDto = new NicknameDto("사용가능한닉네임");
    NicknameResponseDto nicknameResponseDto = new NicknameResponseDto(false);
    ResponseDto<NicknameResponseDto> responseDto = ResponseDto.setSuccess("Nickname not duplicate", nicknameResponseDto);

    given(authService.nickname(nicknameDto)).willReturn(responseDto);

    mockMvc.perform(post("/user/nickname")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(nicknameDto)))
        .andExpect(status().isOk())
        .andExpect(content().string(objectMapper.writeValueAsString(responseDto)));
}

```

#### <whenNicknameNotDuplicate\_thenReturns200 Test>

- 목적: 사용자가 입력한 닉네임이 사용 가능한지 확인한다. 닉네임이 중복되지 않았을 때 적절한 응답이 반환되는지 확인한다.
- 테스트 방법: 모의 AuthService를 통해 닉네임 중복 확인 요청을 시뮬레이션하고, 중복되지 않은 경우 적절한 응답이 반환되는지 확인한다.

```

@DisplayName("아이디 중복") no usages bowoock
@Test
void whenIdDuplicate_thenReturns400() throws Exception {
    IdDto idDto = new IdDto("사용중인아이디");
    IdResponseDto idResponseDto = new IdResponseDto(true);
    ResponseDto<IdResponseDto> responseDto = ResponseDto.setFailed("Id already exists", idResponseDto);

    given(authService.id(idDto)).willReturn(responseDto);

    mockMvc.perform(post("/user/id")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(idDto)))
        .andExpect(status().isOk())
        .andExpect(content().string(objectMapper.writeValueAsString(responseDto)));
}

```

<whenIdDuplicate\_thenReturns400 Test>

- 목적: 사용자가 제공한 아이디가 이미 사용 중인지 확인한다. 아이디가 중복되었을 때 적절한 오류 응답이 반환되는지 확인한다.
- 테스트 방법: 모의 AuthService를 통해 아이디 중복 확인 요청을 시뮬레이션하고, 중복된 경우 적절한 오류 응답이 반환되는지 확인한다.

```

@DisplayName("사용 가능한 아이디") no usages bowoock
@Test
public void whenIdDuplicate_thenReturns200() throws Exception {
    IdDto idDto = new IdDto("사용가능한아이디");
    IdResponseDto idResponseDto = new IdResponseDto(false);
    ResponseDto<IdResponseDto> responseDto = ResponseDto.setSuccess("Id not duplicate", idResponseDto);

    given(authService.id(idDto)).willReturn(responseDto);

    mockMvc.perform(post("/user/id")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(idDto)))
        .andExpect(status().isOk())
        .andExpect(content().string(objectMapper.writeValueAsString(responseDto)));
}

```

<whenIdDuplicate\_thenReturns200 Test>

- 목적: 사용자가 입력한 아이디가 사용 가능한지 확인한다. 아이디가 중복되지 않았을 때 적절한 응답이 반환되는지 확인한다.
- 테스트 방법: 모의 AuthService를 통해 아이디 중복 확인 요청을 시뮬레이션하고, 중복되지 않은 경우 적절한 응답이 반환되는지 확인한다.

```

@DisplayName("유효한 토큰으로 유저 조회") no usages 1 bowook
@Test
void getUserProjects_withValidToken() throws Exception {
    // given
    String validToken = "Bearer 유효한토큰"; // 실제 유효한 토큰으로 대체 필요
    TokenResponseDto tokenResponseDto = new TokenResponseDto(1L, "testLoginId", "testNickname");
    ResponseDto<TokenResponseDto> expectedResponse = ResponseDto.setSuccess("Token valid", tokenResponseDto);

    given(authService.token(anyString())).willReturn(expectedResponse);

    // when & then
    mockMvc.perform(get("/user/check")
        .header("Authorization", validToken))
        .andExpect(status().isOk())
        .andExpect(content().string(objectMapper.writeValueAsString(expectedResponse)));
}

```

### <getUserProjects\_withValidToken Test>-추가 기능

- 목적: 유효한 인증 토큰을 가지고 있는 사용자의 조회가 정상적으로 처리되는지 확인한다. 인증 토큰이 유효한 경우 사용자 정보가 반환되는지 확인한다.
- 테스트 방법: 모의 AuthService를 통해 토큰 검증 요청을 시뮬레이션하고, 유효한 토큰인 경우 사용자 정보가 반환되는지 확인한다.

```

@DisplayName("유효하지 않은 토큰으로 유저 조회 시 오류 반환") no usages 1 bowook
@Test
void getUserProjects_withInvalidToken() throws Exception {
    // given
    String invalidToken = "Bearer 유효하지않은토큰";
    given(authService.token(anyString())).willThrow(new RuntimeException(HttpStatus.UNAUTHORIZED, "Invalid Token"));

    // when & then
    mockMvc.perform(get("/user/check")
        .header("Authorization", invalidToken))
        .andExpect(status().isUnauthorized());
}

```

### <getUserProjects\_withInvalidToken Test>-추가 기능

- 목적: 유효하지 않은 인증 토큰을 가지고 있는 사용자의 조회 시도가 실패하는지 확인한다. 인증 토큰이 유효하지 않은 경우 적절한 오류 응답이 반환되는지 확인한다.
- 테스트 방법: 모의 AuthService를 통해 토큰 검증 요청을 시뮬레이션하고, 유효하지 않은 토큰인 경우 적절한 오류 응답이 반환되는지 확인한다.



## [IssueControllerTest]

```
@Test no usages 1 bowook +1
@DisplayName("이슈 생성 성공")
void issueCreateSuccess() throws Exception {
    // 준비
    long projectId = 1L;
    long issueId = 1L;
    IssueRequestDto.JoinIssueCreateRequestDto requestDto = new IssueRequestDto.JoinIssueCreateRequestDto();
    ReflectionTestUtils.setField(requestDto, "name", "test");
    ReflectionTestUtils.setField(requestDto, "description", "test");
    ReflectionTestUtils.setField(requestDto, "priority", Priority.MAJOR);
    Issue issue = new Issue();
    ReflectionTestUtils.setField(issue, "id", issueId); // Issue의 id 필드에 접근하여 값을 설정
    String jwtToken = "Bearer test.jwt.token";

    when(tokenProvider.validateJwt(anyString())).thenReturn("1"); // 토큰 검증 모의 처리
    when(issueCreateService.joinIssue(any(IssueRequestDto.JoinIssueCreateRequestDto.class), anyLong())).thenReturn(issue); // 이슈

    // 실행 & 검증
    mockMvc.perform(post("/issue/create/{id}", projectId)
        .header("Authorization", jwtToken)
        .contentType(MediaType.APPLICATION_JSON)
        .content(new ObjectMapper().writeValueAsString(requestDto)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.result.id").value(issueId));
}
```

### <issueCreateSuccess Test>

- 목적: 이슈 생성이 정상적으로 처리되는지 확인한다.
- 테스트 방법: 가짜 IssueCreateService를 모의하고, 요청된 데이터로 이슈가 생성되는지 확인한다.

```
@Test no usages 1 bowook
@DisplayName("이슈 삭제 성공")
void issueDeleteSuccess() throws Exception {
    long issueId = 1L;
    String jwtToken = "Bearer test.jwt.token";

    when(tokenProvider.validateJwt(anyString())).thenReturn("1"); // 토큰 검증 모의 처리

    mockMvc.perform(delete("/issue/delete/{id}", issueId)
        .header("Authorization", jwtToken)
        .contentType(MediaType.APPLICATION_JSON)
        .andDo(print()))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.result.id").value(issueId));

    verify(issueDeleteService).deleteIssue(issueId);
}
```

### <issueDeleteSuccess Test>

- 목적: 이슈 삭제가 정상적으로 처리되는지 확인한다.
- 테스트 방법: 가짜 IssueDeleteService를 모의하고, 요청된 이슈가 삭제되는지 확인한다.



```

@Test no usages 1 bowook +1
@DisplayName("이슈 목록 조회 성공 - 이름과 상태로 검색")
void searchIssuesByNameAndStatusSuccess() throws Exception {
    long projectId = 1L;
    String name = "이슈";
    String status = "OPEN"; // 예시 상태 값
    String jwtToken = "Bearer test.jwt.token";

    List<IssueResponseDto.JoinIssueSearchResponseDto> responseDtoList = List.of(
        IssueResponseDto.JoinIssueSearchResponseDto.builder()
            .id(1L)
            .name("이슈1")
            .description("이슈 설명1")
            .status(Status.NEW) // 예시 상태 값
            .projectId(projectId)
            .priority(Priority.BLOCKER)
            .modDate("2023-06-01")
            .build()
    );

    when(tokenProvider.validateJwt(anyString())).thenReturn("1"); // 토큰 검증 모의 처리
    when(issueSearchService.searchIssuesByProjectIdNameAndStatus(projectId, name, status)).thenReturn(responseDtoList);

    mockMvc.perform(get("/issue/search/{id}", projectId)
        .header("Authorization", jwtToken)
        .param("name", name)
        .param("status", status))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$[0].id").value(1L))
        .andExpect(jsonPath("$[0].name").value("이슈1"))
        .andExpect(jsonPath("$[0].status").value("NEW"));
}

```

#### <searchIssuesByNameAndStatusSuccess Test>

- 목적: 이름과 상태로 이슈를 검색하는 기능이 정상적으로 작동하는지 확인한다.
- 테스트 방법: 가짜 IssueSearchService를 모의하고, 요청된 이름과 상태에 해당하는 이슈 목록이 반환되는지 확인한다.

```

@Test no usages 1 bowook
@DisplayName("코멘트 생성 성공")
void createCommentSuccess() throws Exception {
    // 준비
    long issueId = 1L;
    long commentId = 1L;
    String jwtToken = "Bearer test.jwt.token";

    IssueRequestDto.JoinCommentCreateRequestDto requestDto = new IssueRequestDto.JoinCommentCreateRequestDto();
    ReflectionTestUtils.setField(requestDto, "comment", "This is a test comment");
    ReflectionTestUtils.setField(requestDto, "tag", CommentTag.ISSUE_CREATED);

    Comment comment = new Comment();
    ReflectionTestUtils.setField(comment, "id", commentId);

    when(tokenProvider.validateJwt(anyString())).thenReturn("1"); // 토큰 검증 모의 처리
    when(commentCreateService.createComment(any(IssueRequestDto.JoinCommentCreateRequestDto.class), eq(issueId), anyString())).thenReturn(comment);

    // 실행 & 검증
    mockMvc.perform(post("/issue/comment/create/{id}", issueId)
        .header("Authorization", jwtToken)
        .contentType(MediaType.APPLICATION_JSON)
        .content(new ObjectMapper().writeValueAsString(requestDto)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.result.id").value(commentId));
}

```

#### <createCommentSuccess Test>

- 목적: 특정 이슈에 새로운 코멘트를 생성하는 기능이 정상적으로 작동하는지 확인한다.
- 테스트 방법: 가짜 CommentCreateService를 모의하고, 요청된 코멘트가 생성되는지 확인한다.

```

@Test no usages  ⚡ bowook +1
@DisplayName("이슈 목록 조회 성공 - 프로젝트 ID로만 검색")
void searchIssuesByProjectIdSuccess() throws Exception {
    long projectId = 1L;
    String jwtToken = "Bearer test.jwt.token";

    List<IssueResponseDto, JoinIssueSearchResponseDto> responseDtoList = List.of(
        IssueResponseDto, JoinIssueSearchResponseDto.builder()
            .id(1L)
            .name("이슈1")
            .description("이슈 설명1")
            .status(Status.NEW) // 예시 상태 값
            .priority(Priority.CRITICAL)
            .projectId(projectId)
            .modDate("2023-06-01")
            .build()
    );

    when(tokenProvider.validateJwt(anyString())).thenReturn("1"); // 토큰 검증 모의 처리
    when(issueSearchService.searchIssuesByProjectId(projectId)).thenReturn(responseDtoList);

    mockMvc.perform(get("/issue/search/{id}", projectId)
        .header("Authorization", jwtToken))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.id").value(1L))
        .andExpect(jsonPath("$.name").value("이슈1"))
        .andExpect(jsonPath("$.status").value("NEW"));
}

```

#### <searchIssuesByProjectIdSuccess Test>

- 목적: 프로젝트 ID로 이슈를 검색하는 기능이 정상적으로 작동하는지 확인한다.
- 테스트 방법: 가짜 IssueSearchService를 모의하고, 요청된 프로젝트 ID에 해당하는 이슈 목록이 반환되는지 확인한다.

```

@Test no usages bowook +1
@DisplayName("이슈 정보 조회 성공")
void issueInfoSuccess() throws Exception {
    long issueId = 1L;
    String jwtToken = "Bearer test.jwt.token";

    IssueResponseDto.JoinIssueInfoResponseDto responseDto = IssueResponseDto.JoinIssueInfoResponseDto.builder()
        .id(issueId)
        .name("테스트 이슈")
        .description("테스트 설명")
        .status(Status.NEW)
        .priority(Priority.TRIVIAL)
        .projectId(1L)
        .initDate("2023-06-01")
        .modDate("2023-06-01")
        .assignee("테스트 담당자")
        .reporter("테스트 보고자")
        .build();

    when(tokenProvider.validateJwt(anyString())).thenReturn("1"); // 토큰 검증 모의 처리
    when(issueInfoService.getIssueDetails(issueId)).thenReturn(responseDto); // 이슈 정보 서비스 호출 결과 모의 처리

    // 실행 & 검증
    mockMvc.perform(get("/issue/info/{id}", issueId)
        .header("Authorization", jwtToken)
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.result.id").value(issueId))
        .andExpect(jsonPath("$.result.name").value("테스트 이슈"))
        .andExpect(jsonPath("$.result.description").value("테스트 설명"))
        .andExpect(jsonPath("$.result.status").value("NEW"))
        .andExpect(jsonPath("$.result.priority").value("TRIVIAL"))
        .andExpect(jsonPath("$.result.projectId").value(1L))
        .andExpect(jsonPath("$.result.initDate").value("2023-06-01"))
        .andExpect(jsonPath("$.result.modDate").value("2023-06-01"))
        .andExpect(jsonPath("$.result.assignee").value("테스트 담당자"))
        .andExpect(jsonPath("$.result.reporter").value("테스트 보고자"));
}

```

#### <issueInfoSuccess Test>

- 목적: 특정 이슈의 정보를 조회하는 기능이 정상적으로 작동하는지 확인한다.
- 테스트 방법: 가짜 IssueInfoServiceImpl를 모의하고, 요청된 이슈 정보가 반환되는지 확인한다.

```

@Test no usages bowook
@DisplayName("이슈 상태 변경 성공")
void changeIssueStateSuccess() throws Exception {
    // 준비
    long issueId = 1L; // 테스트 대상 이슈 ID
    Status newStatus = Status.FIXED; // 변경할 새 상태
    IssueRequestDto.JoinIssueStateRequestDto requestDto = new IssueRequestDto.JoinIssueStateRequestDto();
    ReflectionTestUtils.setField(requestDto, "status", newStatus); // 변경할 상태로 수정, 즉, 변경할 상태를 전달받음

    //서비스에서 상태를 return하는게 없기 때문에, 상태 변경 서비스가 동작하는지만 확인해야함.

    String jwtToken = "Bearer test.jwt.token";

    when(tokenProvider.validateJwt(anyString())).thenReturn("1"); // 토큰 검증 모의 처리

    // 실행 & 검증
    mockMvc.perform(post("/issue/state/{id}", issueId)
        .header("Authorization", jwtToken)
        .contentType(MediaType.APPLICATION_JSON)
        .content(new ObjectMapper().writeValueAsString(requestDto)))
        .andExpect(status().isOk());

    // 이슈 상태 변경 서비스가 올바르게 호출되었는지 확인
    verify(issueStateService).changeIssueState(any(IssueRequestDto.JoinIssueStateRequestDto.class), eq(issueId));
}

```

#### <changeIssueStateSuccess Test>

- 목적: 특정 이슈의 상태를 변경하는 기능이 정상적으로 작동하는지 확인한다.
- 테스트 방법: 가짜 IssueStateService를 모의하고, 요청된 상태로 이슈의 상태가 변경되는지 확인한다.

```

@Test no usages bowook
@DisplayName("담당자 할당 성공")
void assignAssigneeSuccess() throws Exception {
    // 준비
    long issueId = 1L; // 테스트에 사용할 이슈 ID
    String nickname = "testUser"; // 테스트에 사용할 담당자 닉네임
    IssueRequestDto.JoinAssigneeRequestDto requestDto = new IssueRequestDto.JoinAssigneeRequestDto();
    ReflectionTestUtils.setField(requestDto, "nickname", nickname); // Request DTO에 닉네임 설정

    Assignee assignee = new Assignee(); // Assignee 엔티티 모의 객체
    User user = new User();
    ReflectionTestUtils.setField(user, "nickname", nickname); // User에 닉네임 설정
    ReflectionTestUtils.setField(assignee, "user", user); // Assignee에 User 설정

    when(setAssigneeService.assignAssignee(eq(issueId), any(IssueRequestDto.JoinAssigneeRequestDto.class))).thenReturn(assignee);

    // 실행 & 검증
    mockMvc.perform(post("/issue/assignee/{id}", issueId)
        .contentType(MediaType.APPLICATION_JSON)
        .content(new ObjectMapper().writeValueAsString(requestDto)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.result.nickname").value(nickname)); // 응답에서 닉네임 검증
}

```

#### <assignAssigneeSuccess Test>-추가 기능

- 목적: 특정 이슈에 담당자를 할당하는 기능이 정상적으로 작동하는지 확인한다.
- 테스트 방법: 가짜 SetAssigneeService를 모의하고, 요청된 담당자가 할당되는지 확인한다.

```

@Test no usages ⚡ bowook
@DisplayName("특정 이슈의 댓글 목록 조회 성공")
void getCommentsByIssueIdSuccess() throws Exception {
    long issueId = 1L;
    List<IssueResponseDto.GetCommentResponseDto> comments = List.of(
        IssueResponseDto.GetCommentResponseDto.builder()
            .id(1L)
            .comment("Test comment 1")
            .tag(CommentTag.ISSUE_CREATED)
            .nickname("user1")
            .modDate("2023-06-01T10:00:00")
            .build(),
        IssueResponseDto.GetCommentResponseDto.builder()
            .id(2L)
            .comment("Test comment 2")
            .tag(CommentTag.SET_FIXED)
            .nickname("user2")
            .modDate("2023-06-01T11:00:00")
            .build()
    );

    when(commentListService.getCommentsByIssueId(issueId)).thenReturn(comments);

    mockMvc.perform(get("/issue/comment/{id}", issueId)
        .contentType(MediaType.APPLICATION_JSON))
        .andDo(print())
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.length()").value(comments.size()))
        .andExpect(jsonPath("$[0].id").value(comments.get(0).getId()))
        .andExpect(jsonPath("$[0].comment").value(comments.get(0).getComment()))
        .andExpect(jsonPath("$[0].tag").value(comments.get(0).getTag().toString()))
        .andExpect(jsonPath("$[0].nickname").value(comments.get(0).getNickname()))
        .andExpect(jsonPath("$[0].modDate").value(comments.get(0).getModDate()))
        .andExpect(jsonPath("$[1].id").value(comments.get(1).getId()))
        .andExpect(jsonPath("$[1].comment").value(comments.get(1).getComment()))
        .andExpect(jsonPath("$[1].tag").value(comments.get(1).getTag().toString()))
        .andExpect(jsonPath("$[1].nickname").value(comments.get(1).getNickname()))
        .andExpect(jsonPath("$[1].modDate").value(comments.get(1).getModDate()));
}

```

#### <getCommentsByIssueIdSuccess Test>

- 목적: 이슈에 대한 댓글 목록을 조회하는 기능이 정상적으로 작동하는지 확인한다.
- 테스트 방법: 가짜 CommentListService를 모의하고, 요청된 이슈의 댓글 목록이 반환되는지 확인한다.



```

@Test no usages  bowook
@DisplayName("리포터 할당 성공")
void assignReporterToIssueSuccess() throws Exception {
    // 준비
    long issueId = 1L; // 테스트를 위한 이슈 ID
    String nickname = "testReporter"; // 할당할 리포터의 닉네임

    IssueRequestDto.JoinAssigneeRequestDto requestDto = new IssueRequestDto.JoinAssigneeRequestDto();
    ReflectionTestUtils.setField(requestDto, "nickname", nickname); // 리포터의 닉네임 설정

    Reporter reporter = new Reporter(); // Reporter 엔티티 생성 및 설정
    // Reporter 엔티티에 필요한 필드 설정 생략
    User user = new User(); // User 객체 생성 및 설정
    ReflectionTestUtils.setField(user, "nickname", nickname); // User에 닉네임 설정
    ReflectionTestUtils.setField(reporter, "user", user); // Reporter에 User 설정

    when(setAssigneeService.assignReporter(eq(issueId), any(IssueRequestDto.JoinAssigneeRequestDto.class)))
        .thenReturn(reporter); // assignReporter 호출 결과 모의 처리

    String jwtToken = "Bearer test.jwt.token"; // 테스트용 JWT 토큰

    // 실행 & 검증
    mockMvc.perform(post("/issue/reporter/{id}", issueId)
        .header("Authorization", jwtToken)
        .contentType(MediaType.APPLICATION_JSON)
        .content(new ObjectMapper().writeValueAsString(requestDto)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.result.nickname").value(nickname));
}

```

#### <assignReporterToIssueSuccess Test>

- 목적: 특정 이슈에 리포터를 할당하는 기능이 정상적으로 작동하는지 확인한다.
- 테스트 방법: 가짜 SetAssigneeService를 모의하고, 요청된 리포터가 할당되는지 확인한다.

```

@Test
@DisplayName("가장 적은 이슈가 할당된 개발자 목록 가져오기 성공")
void getDevelopersWithLeastAssignmentsSuccess() throws Exception {
    // 준비
    long projectId = 1L;
    List<IssueResponseDto.AssigneeAlgoResponseDto> developers = new ArrayList<>();
    IssueResponseDto.AssigneeAlgoResponseDto developer = new IssueResponseDto.AssigneeAlgoResponseDto();
    developer.setId(1L);
    developer.setNickname("dev1");
    developer.setRole(Role.DEVELOPER);
    developer.setAssignedIssueCount(2L);
    developers.add(developer);

    when(assigneeAlgoService.getDevelopersWithLeastAssignments(projectId)).thenReturn(developers); // 서비스 호출 결과 모의 처리

    // 실행 & 검증
    mockMvc.perform(get(uriTemplate: "/issue/assignee/algo/{id}", projectId)
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath("expression: \"${0}.id\"").value(developer.getId()))
        .andExpect(jsonPath("expression: \"${0}.nickname\"").value(developer.getNickname()))
        .andExpect(jsonPath("expression: \"${0}.role\"").value(developer.getRole().toString()))
        .andExpect(jsonPath("expression: \"${0}.assignedIssueCount\"").value(developer.getAssignedIssueCount()));
}

```

#### <getDevelopersWithLeastAssignmentSuccess Test>

- 목적: 이슈 관리 시스템에서 가장 적은 이슈가 할당된 개발자 목록을 가져오는 요청에 대해 올바른 응답을 반환하는지 확인한다.
- 테스트 방법: MockMvc를 사용하여 특정 프로젝트 ID에 대해 GET 요청을 보내고, 반환된 개발자 목록의 각 항목이 예상한 대로인지 확인한다.

## [ProjectControllerTest]

```

1 bowook
@Test
@DisplayName("프로젝트 생성 성공")
void projectCreateSuccess() throws Exception {
    long projectId = 1L;
    ProjectRequestDto.JoinProjectCreateRequestDto requestDto = new ProjectRequestDto.JoinProjectCreateRequestDto();
    ReflectionTestUtils.setField(requestDto, "name", "test");
    ReflectionTestUtils.setField(requestDto, "description", "test");
    ReflectionTestUtils.setField(requestDto, "isPrivate", true);

    Project project = new Project();

    ReflectionTestUtils.setField(project, "id", projectId); // project id 필드에 접근하여 값을 설정

    String jwtToken = "Bearer test.jwt.token";

    when(tokenProvider.validateJwt(anyString())).thenReturn("1"); // 토큰 검증 모의 처리
    when(projectCreateService.joinProject(any(ProjectRequestDto.JoinProjectCreateRequestDto.class))).thenReturn(project); // 프로젝트 생성 서비스 호출 결과 모의 처리

    // 실행 & 검증
    mockMvc.perform(post("/project/create")
        .header("Authorization", jwtToken)
        .contentType(MediaType.APPLICATION_JSON)
        .content(new ObjectMapper().writeValueAsString(requestDto)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.result.id").value(projectId));
}

```

### <projectCreateSuccess Test>

- 목적: 프로젝트 생성 요청이 제대로 처리되고 새로운 프로젝트가 생성되는지 확인한다.
- 테스트 방법: Mock를 사용하여 POST 요청을 보내고, 생성된 프로젝트의 ID를 확인한다.

```

1 bowook
@Test
@DisplayName("프로젝트 팀 가입 성공")
void joinTeamSuccess() throws Exception {
    long projectId = 1L;
    ProjectRequestDto.JoinTeamRequestDto requestDto = new ProjectRequestDto.JoinTeamRequestDto();
    ReflectionTestUtils.setField(requestDto, "nickname", "userNickname");
    ReflectionTestUtils.setField(requestDto, "isAdmin", false);
    ReflectionTestUtils.setField(requestDto, "role", Role.DEVELOPER);

    Team team = new Team();
    ReflectionTestUtils.setField(team, "id", projectId); // team id 필드에 접근하여 값을 설정

    String jwtToken = "Bearer test.jwt.token";

    when(tokenProvider.validateJwt(anyString())).thenReturn("1"); // 토큰 검증 모의 처리
    when(teamJoinService.joinTeam(any(ProjectRequestDto.JoinTeamRequestDto.class), eq(projectId))).thenReturn(team); // 팀 가입 서비스 호출 결과 모의 처리

    // 실행 & 검증
    mockMvc.perform(post("/project/team/{id}", projectId)
        .header("Authorization", jwtToken)
        .contentType(MediaType.APPLICATION_JSON)
        .content(new ObjectMapper().writeValueAsString(requestDto)))
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.result.id").value(projectId));
}

```

### <joinTeamSuccess Test>

- 목적: 사용자가 프로젝트 팀에 가입할 수 있는지 확인한다.
- 테스트 방법: Mock를 사용하여 POST 요청을 보내고, 팀 가입이 올바르게 처리되는지 확인한다.

```

@bawook
@Test
@DisplayName("프로젝트 팀 멤버 조회 성공")
void getTeamsByProject() throws Exception {
    long projectId = 1L;

    List<ProjectResponseDto.TeamMemberDTO> teamMembers = Arrays.asList(
        new ProjectResponseDto.TeamMemberDTO( isAdmin: true, Role.PL, nickname: "plNickname"),
        new ProjectResponseDto.TeamMemberDTO( isAdmin: false, Role.DEVELOPER, nickname: "developerNickname")
    );

    when(teamMemberService.getTeamsByProjectId(projectId)).thenReturn(ResponseEntity.ok(teamMembers));

    String jwtToken = "Bearer test.jwt.token";

    // 실행 & 검증
    mockMvc.perform(get( uriTemplate: "/project/team/member/{id}", projectId)
        .header( name: "Authorization", jwtToken)
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath( expression: "$[0].isAdmin").value( expectedValue: true))
        .andExpect(jsonPath( expression: "$[0].role").value( expectedValue: "PL"))
        .andExpect(jsonPath( expression: "$[0].nickname").value( expectedValue: "plNickname"))
        .andExpect(jsonPath( expression: "$[1].isAdmin").value( expectedValue: false))
        .andExpect(jsonPath( expression: "$[1].role").value( expectedValue: "DEVELOPER"))
        .andExpect(jsonPath( expression: "$[1].nickname").value( expectedValue: "developerNickname"));
}

```

#### <getTeamByProject Test>

- 목적: 특정 프로젝트의 팀 멤버들을 조회할 수 있는지 확인한다.
- 테스트 방법: MockMvc를 사용하여 GET 요청을 보내고, 반환된 팀 멤버 목록이 예상대로 인지 확인한다.

```

@bawook
@Test
@DisplayName("프로젝트 이름으로 검색 성공")
void searchProjectsByNameSuccess() throws Exception {
    List<Project> projects = Arrays.asList(new Project(), new Project()); // 가정한 프로젝트 목록
    String searchName = "testProject";

    when(projectSearchService.searchProjectsByName(searchName)).thenReturn(projects); // 이름으로 프로젝트 검색 시 모의 처리

    mockMvc.perform(get( uriTemplate: "/project/search")
        .param( name: "name", searchName))
        .andExpect(status().isOk())
        .andExpect(jsonPath( expression: "$.length()").value(projects.size())); // 반환된 프로젝트 목록의 크기 검증
}

```

#### <searchProjectsByNameSuccess Test>



- 목적: 특정 이름을 가진 프로젝트들을 검색할 수 있는지 확인한다.
- 테스트 방법: MockMvc를 사용하여 GET 요청을 보내고, 검색 결과가 예상대로인지 확인한다.

```

± bowook
@Test
@DisplayName("모든 프로젝트 검색 성공")
void searchAllProjectsSuccess() throws Exception {
    List<Project> projects = Arrays.asList(new Project(), new Project(), new Project()); // 가정한 프로젝트 목록

    when(projectSearchService.searchAllProjects()).thenReturn(projects); // 모든 프로젝트 검색 시 모의 처리

    mockMvc.perform(get(uriTemplate: "/project/search"))
        .andExpect(status().isOk())
        .andExpect(jsonPath(expression: "$.length").value(projects.size())); // 반환된 프로젝트 목록의 크기 검증
}

```

#### <searchAllProjectsSuccess Test>

- 목적: 모든 프로젝트를 검색할 수 있는지 확인한다.
- 테스트 방법: MockMvc를 사용하여 GET 요청을 보내고, 검색 결과가 예상대로인지 확인한다.

```

± bowook
@Test
@DisplayName("프로젝트 정보 조회 성공")
void getProjectInfo() throws Exception {
    long projectId = 1L;

    Project project = new Project();
    ReflectionTestUtils.setField(project, name: "id", value: projectId);
    ReflectionTestUtils.setField(project, name: "name", value: "testProject");
    ReflectionTestUtils.setField(project, name: "description", value: "testDescription");

    ApiResponse<Project> response = new ApiResponse<>(< isSuccess: true, code: "200", message: "Project retrieved successfully", project);

    when(projectInfoService.projectInfo(projectId)).thenReturn(project);

    // 실행 & 검증
    mockMvc.perform(get(uriTemplate: "/project/info/{id}", projectId)
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath(expression: "$.isSuccess").value( expectedValue: true))
        .andExpect(jsonPath(expression: "$.code").value( expectedValue: "200"))
        .andExpect(jsonPath(expression: "$.message").value( expectedValue: "Project retrieved successfully"))
        .andExpect(jsonPath(expression: "$.result.id").value(projectId))
        .andExpect(jsonPath(expression: "$.result.name").value( expectedValue: "testProject"))
        .andExpect(jsonPath(expression: "$.result.description").value( expectedValue: "testDescription"));
}

```

#### <getProjectInfo Test>

- 목적: 특정 프로젝트의 정보를 조회할 수 있는지 확인한다.
- 테스트 방법: MockMvc를 사용하여 GET 요청을 보내고, 반환된 프로젝트 정보가 예상대로인지 확인한다.

```

@Test
@DisplayName("사용자 프로젝트 조회 성공")
void getUserProjectsSuccess() throws Exception {
    String jwtToken = "Bearer test.jwt.token";
    String token = "test.jwt.token";
    Long userId = 1L;
    List<ProjectResponseDto.MyProjectResponseDto> projects = Arrays.asList(
        ProjectResponseDto.MyProjectResponseDto.builder()
            .id(1L).name("Project 1")
            .isPrivate(false).description("Description 1")
            .thumbnailLink("http://example.com/image1.png")
            .initdate(LocalDateTime.now().minusDays(5))
            .moddate(LocalDateTime.now().minusDays(3))
            .build(),
        ProjectResponseDto.MyProjectResponseDto.builder()
            .id(2L)
            .name("Project 2")
            .isPrivate(true)
            .description("Description 2")
            .thumbnailLink("http://example.com/image2.png")
            .initdate(LocalDateTime.now().minusDays(10))
            .moddate(LocalDateTime.now().minusDays(1))
            .build()
    );
    when(tokenProvider.validateJwt(anyString())).thenReturn(userId.toString());
    when(myProjectService.getProjectsByUserToken(token)).thenReturn(projects);
    mockMvc.perform(get( uriTemplate: "/project/my")
        .header( name: "Authorization", jwtToken)
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath( expression: "$[0].id").value( expectedValue: 1L))
        .andExpect(jsonPath( expression: "$[0].name").value( expectedValue: "Project 1"))
        .andExpect(jsonPath( expression: "$[0].isPrivate").value( expectedValue: false))
        .andExpect(jsonPath( expression: "$[0].description").value( expectedValue: "Description 1"))
        .andExpect(jsonPath( expression: "$[0].thumbnailLink").value( expectedValue: "http://example.com/image1.png"))
        .andExpect(jsonPath( expression: "$[0].initdate").exists())
        .andExpect(jsonPath( expression: "$[0].moddate").exists())
        .andExpect(jsonPath( expression: "$[1].id").value( expectedValue: 2L))
        .andExpect(jsonPath( expression: "$[1].name").value( expectedValue: "Project 2"))
        .andExpect(jsonPath( expression: "$[1].isPrivate").value( expectedValue: true))
        .andExpect(jsonPath( expression: "$[1].description").value( expectedValue: "Description 2"))
        .andExpect(jsonPath( expression: "$[1].thumbnailLink").value( expectedValue: "http://example.com/image2.png"))
        .andExpect(jsonPath( expression: "$[1].initdate").exists())
        .andExpect(jsonPath( expression: "$[1].moddate").exists());
}

```

- 목적: 특정 사용자의 프로젝트를 조회할 수 있는지 확인한다.
- 테스트 방법: MockMvc를 사용하여 GET 요청을 보내고, 반환된 프로젝트 목록이 예상대로인지 확인한다.

```

@bawook
@Test
@DisplayName("프로젝트에서의 역할 조회 성공")
void getMyRoleInProjectSuccess() throws Exception {
    long projectId = 1L;
    long userId = 1L;
    Role role = Role.DEVELOPER; // 직접 enum 값을 사용

    String jwtToken = "Bearer test.jwt.token";

    when(tokenProvider.validateJwt(anyString())).thenReturn(String.valueOf(userId)); // 토큰 검증 모의 처리
    when(myRoleService.findUserRoleInProject(projectId, userId)).thenReturn(role); // 역할 조회 서비스 호출 결과 모의 처리

    // 실행 & 검증
    mockMvc.perform(get( uriTemplate: "/project/role/{id}", projectId)
        .header( name: "Authorization", jwtToken)
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk())
        .andExpect(jsonPath( expression: "$.isSuccess").value( expectedValue: true))
        .andExpect(jsonPath( expression: "$.code").value( expectedValue: "200"))
        .andExpect(jsonPath( expression: "$.message").value( expectedValue: "Role retrieved successfully"))
        .andExpect(jsonPath( expression: "$.result").value(role.name())); // enum의 이름을 문자열로 비교
}

```

<getMyRoleInProjectSuccess Test>

- 목적: 특정 프로젝트에서 사용자의 역할을 조회할 수 있는지 확인한다.
- 테스트 방법: MockMvc를 사용하여 GET 요청을 보내고, 반환된 역할이 예상대로인지 확인한다.

#### ※ 테스트 목적

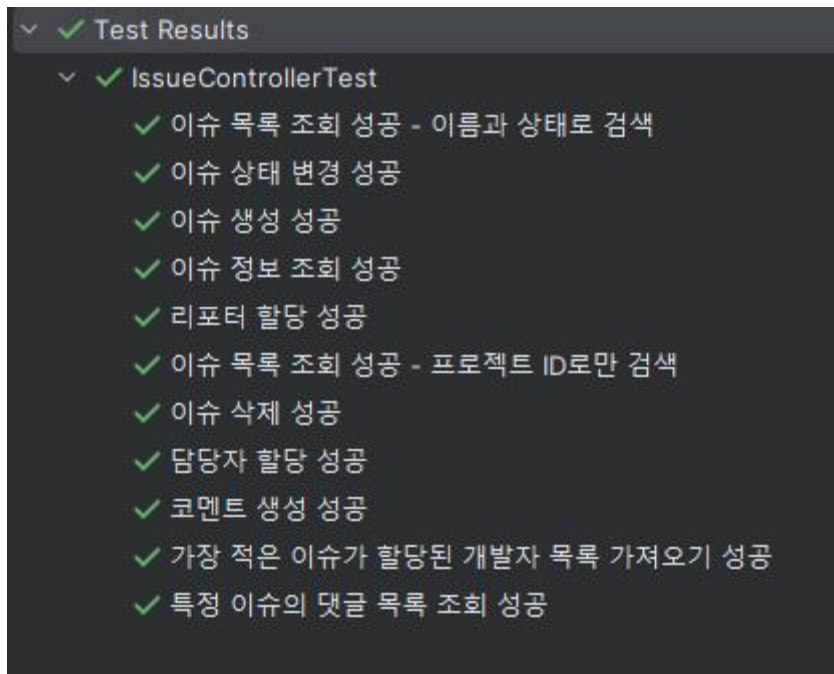
- i) 코드에 잠재된 문제를 미리 발견할 수 있다.
- ii) 애플리케이션을 실행해서 테스트하는 것보다 빠르게 진행할 수 있다.
- iii) 테스트 코드가 작성된 목적을 명확하게 표현할 수 있다.
- iv) 오류가 발생할 수 있는 테스트 코드를 작성해서 예외 처리가 잘 작동하는지 확인하기 위함이다. 다시 말해 의도한 로직에 맞추어 결과 값이 잘 나오는지 검토하기 위함이다.

#### ※ 테스트 도구: JUnit 5, Mockito

### 5.1.1. Test Case 결과

✓ Test Results	2 sec 143 ms
✓ AuthControllerTest	2 sec 143 ms
✓ 사용 가능한 닉네임	1 sec 984 ms
✓ 닉네임 중복	21 ms
✓ 회원 가입 성공	20 ms
✓ 사용 가능한 아이디	19 ms
✓ 아이디 중복	16 ms
✓ 유효한 토큰으로 유저 조회	22 ms
✓ 유효하지 않은 토큰으로 유저 조회 시 오류	20 ms
✓ 로그인 성공	41 ms

<AuthControllerTest result>



<IssueControllerTest result>

Test Results	354 ms
ProjectControllerTest	354 ms
모든 프로젝트 검색 성공	202 ms
프로젝트 정보 조회 성공	22 ms
프로젝트 팀 가입 성공	63 ms
프로젝트 생성 성공	13 ms
프로젝트 팀 멤버 조회 성공	14 ms
프로젝트 이름으로 검색 성공	16 ms
사용자 프로젝트 조회 성공	13 ms
프로젝트에서의 역할 조회 성공	11 ms

<ProjectControllerTest> result>

## 6. GitHub Project 활용

### 6.1 GitHub Project Address of IssueStation

※github Issue-Station 주소

<https://github.com/SE-Issue-Mgmt-Sys>

※github Issue-Station Back-end 주소

<https://github.com/SE-Issue-Mgmt-Sys/Issue-Station-BE.git>

※github Issue-Station GUI 주소

<https://github.com/SE-Issue-Mgmt-Sys/GUI.git>

※github Issue-Station Web 주소

<https://github.com/SE-Issue-Mgmt-Sys/Issue-Station-Web.git>



merge pull request #42 from SE-Issue-Mgmt-Sys/BK	Verified	4598022		<>
project Get method test OK		99014f4		<>
Merge pull request #41 from SE-Issue-Mgmt-Sys/BK	Verified	73c3684		<>
User Testcode		d485978		<>
풀 리퀘스트 병합 #40	Verified	2467b49		<>
issue Info API 수정		904f020		<>
Commits on May 31, 2024				
풀 리퀘스트 병합 #39	Verified	217f2d4		<>
issue Info API 수정		fa2ac09		<>
풀 리퀘스트 병합 #38	Verified	85b0173		<>
issue Info API 수정		7529eed		<>
풀 리퀘스트 병합 #37	Verified	f830a2f		<>
issue Info API 수정		5b9fdeed		<>
Merge pull request #36 from SE-Issue-Mgmt-Sys/PROME	Verified	e0a9a58		<>

## ※github Issue-Station GUI

Commits on Jun 1, 2024

프로젝트 정보 조회도 완료

jeoneunjin committed 2 days ago

dddefc5a

<>

Commits on May 31, 2024

프로젝트 생성,조회,검색 완료

jeoneunjin committed 3 days ago

f58b272

<>

Commits on May 30, 2024

회원가입 로그인 완료

jeoneunjin committed 4 days ago

cdbcdf1

<>

회원가입은 된듯!

jeoneunjin committed 4 days ago

a9ef125

<>

Commits on May 28, 2024

proj detail screen 수정 중

jeoneunjin committed last week

16fabe6

<>

project detail 화면만 하면 대충 끝

jeoneunjin committed last week

b7418f8

<>

Commits on May 26, 2024

project 화면 수정

jeoneunjin committed last week

64d3aa1

<>

Commits

main

All users

All time

Commits on Jun 3, 2024

정말 최종 실행파일

jeoneunjin committed 10 hours ago

7c82158

<>

Update README.md

jeoneunjin committed 10 hours ago

Verified

4d8ebb4

<>

Commits on Jun 2, 2024

Create README.md

jeoneunjin committed 17 hours ago

Verified

2db79a6

<>

최종본 실행파일

jeoneunjin committed 18 hours ago

f48586

<>

dev 배정 추가

jeoneunjin committed 18 hours ago

bc4a855

<>

Delete README.md

jeoneunjin committed 18 hours ago

Verified

6a8d6ca

<>

Update README.md

jeoneunjin committed yesterday

Verified

cd60fa1

<>

Create README.md

jeoneunjin committed yesterday

Verified

8d1d893

<>

최종 실행 파일

jeoneunjin committed yesterday

51d2653

<>

실행 파일 생성

jeoneunjin committed yesterday

16c77f5








<>

이슈 검색,comment 관련 기능, dev 배정 빼고 완료

48f261c

<>



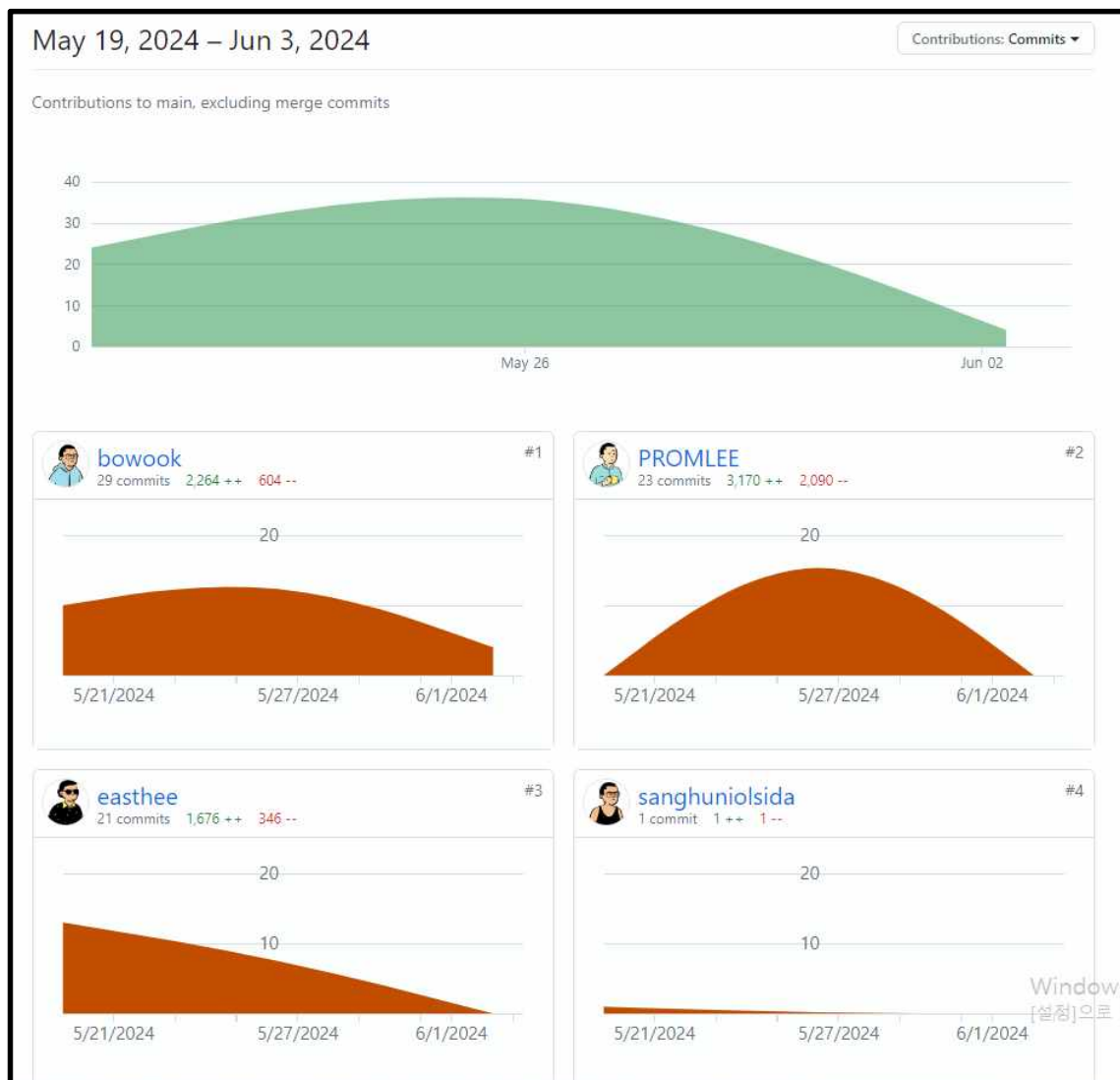
Commits on Jun 1, 2024		
프로젝트 정보 조회도 완료 jeoneunjin committed 2 days ago	ddefc5a	 <a href="#">view</a>
Commits on May 31, 2024		
프로젝트 생성,조회,검색 완료 jeoneunjin committed 3 days ago	f58b272	 <a href="#">view</a>
Commits on May 30, 2024		
회원가입 로그인 완료 jeoneunjin committed 4 days ago	cbcdf1	 <a href="#">view</a>
회원가입은 된듯! jeoneunjin committed 4 days ago	a9af125	 <a href="#">view</a>
Commits on May 28, 2024		
proj detail screen 수정 중 jeoneunjin committed last week	16fabe6	 <a href="#">view</a>
project detail 화면만 하면 대중 끝 jeoneunjin committed last week	b7419f8	 <a href="#">view</a>
Commits on May 26, 2024		
project 화면 수정 jeoneunjin committed last week	64d3aa1	 <a href="#">view</a>

## ※github Issue-Station Web

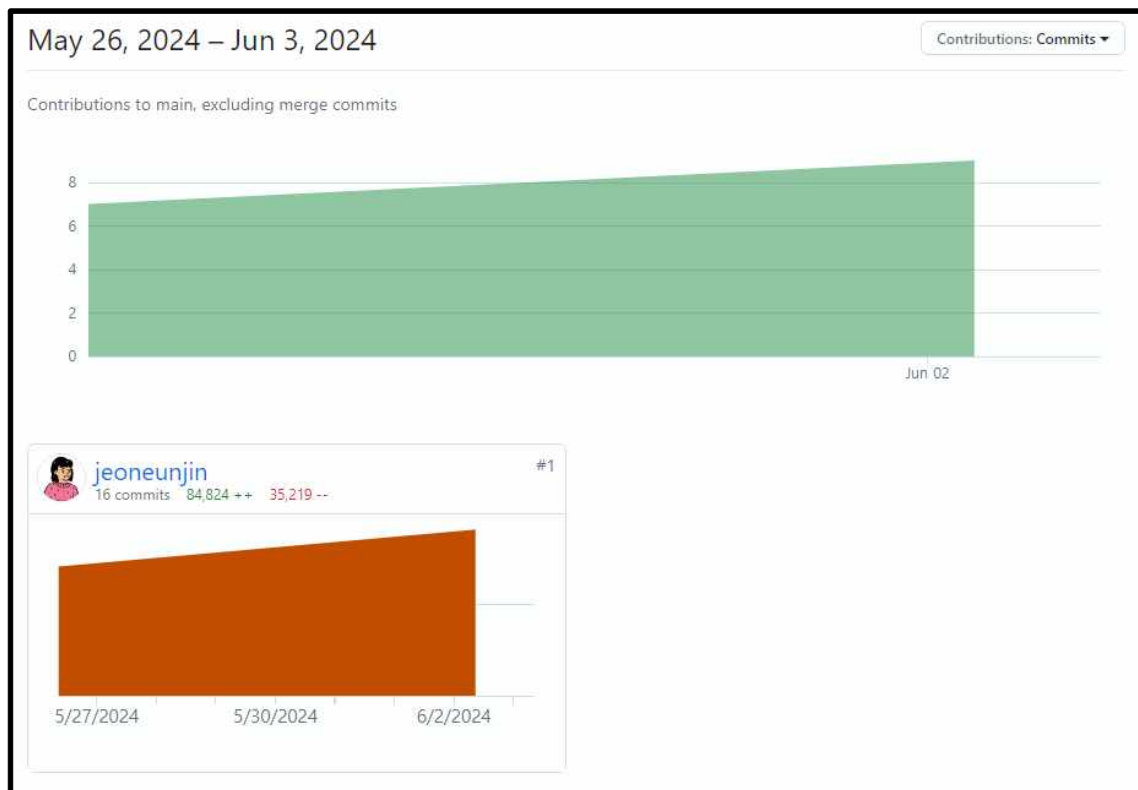
Commits			
main	All users	All time	
Commits on Jun 3, 2024			
Merge pull request #7 from SE-Issue-Mgmt-Sys:dev	Verified	b4b5263	<>
PROMLEE committed 54 minutes ago · ✓ 1 / 1			
Fix duplicate alert message in SignupComponent		62e666a	<>
PROMLEE committed 54 minutes ago			
Commits on Jun 1, 2024			
Merge pull request #6 from SE-Issue-Mgmt-Sys:dev	Verified	9c50eba	<>
PROMLEE committed 3 days ago · ✓ 1 / 1			
feat: Fix duplicate alert issue in SignupComponent		752a610	<>
PROMLEE committed 3 days ago			
Commits on May 31, 2024			
Merge pull request #5 from SE-Issue-Mgmt-Sys:dev	Verified	ab29a52	<>
PROMLEE committed 4 days ago · ✓ 1 / 1			
Refactor TagList object keys to use uppercase and fix typo in Tag component		f86244d	<>
PROMLEE committed 4 days ago			
Commits on May 30, 2024			
feat: Update project and issue pages layout		1b2381c	<>
PROMLEE committed 2 weeks ago			
feat: Remove unused components and assets		a60b8e8	<>
PROMLEE committed 2 weeks ago · ✓ 1 / 1			
Commits on May 17, 2024			
Merge pull request #1 from SE-Issue-Mgmt-Sys:dev	Verified	06973a6	<>
PROMLEE committed 2 weeks ago · ✓ 1 / 1			
Update page title and logo in index.html and Navbar.jsx		7745612	<>
PROMLEE committed 2 weeks ago			
[Feat] create CICD		811032f	<>
PROMLEE committed 2 weeks ago · ✓ 1 / 1			
[Feat]: projectlist, Issuelist, project 검색 화면 구성		088bf83	<>
PROMLEE committed 2 weeks ago			
Commits on May 14, 2024			
feat: Add lazy loading for images and update fonts		0913f65	<>
PROMLEE committed 3 weeks ago			
[Feat] Init		55f32e0	<>
PROMLEE committed 3 weeks ago			
Commits on May 13, 2024			
feat: Refactor IssueCreate component and handle issue creation		88d76ba	<>
PROMLEE committed 4 days ago			
Merge pull request #4 from SE-Issue-Mgmt-Sys:dev	Verified	c81c0fe	<>
PROMLEE committed 4 days ago · ✓ 1 / 1			
build: Update build script to disable CI flag		336bbf0	<>
PROMLEE committed 4 days ago			
Merge pull request #3 from SE-Issue-Mgmt-Sys:dev	Verified	e27a3d0	<>
PROMLEE committed 4 days ago · ✗ 0 / 1			
feat: Refactor CreateProject component and add user list rendering		406e8e0	<>
PROMLEE committed 4 days ago			
feat: Fix duplicate alert issue in SignupComponent		b05f69a	<>
PROMLEE committed 4 days ago			
feat: Remove unused components and assets		8312a08	<>
PROMLEE committed 4 days ago			
Merge pull request #2 from SE-Issue-Mgmt-Sys:dev	Verified	d24805d	<>
PROMLEE committed 4 days ago · ✓ 1 / 1			
feat: Add lazy loading for images and Recoil state management		97457b6	<>
PROMLEE committed 4 days ago			
Commits on May 21, 2024			

## 6.3 팀원 역할

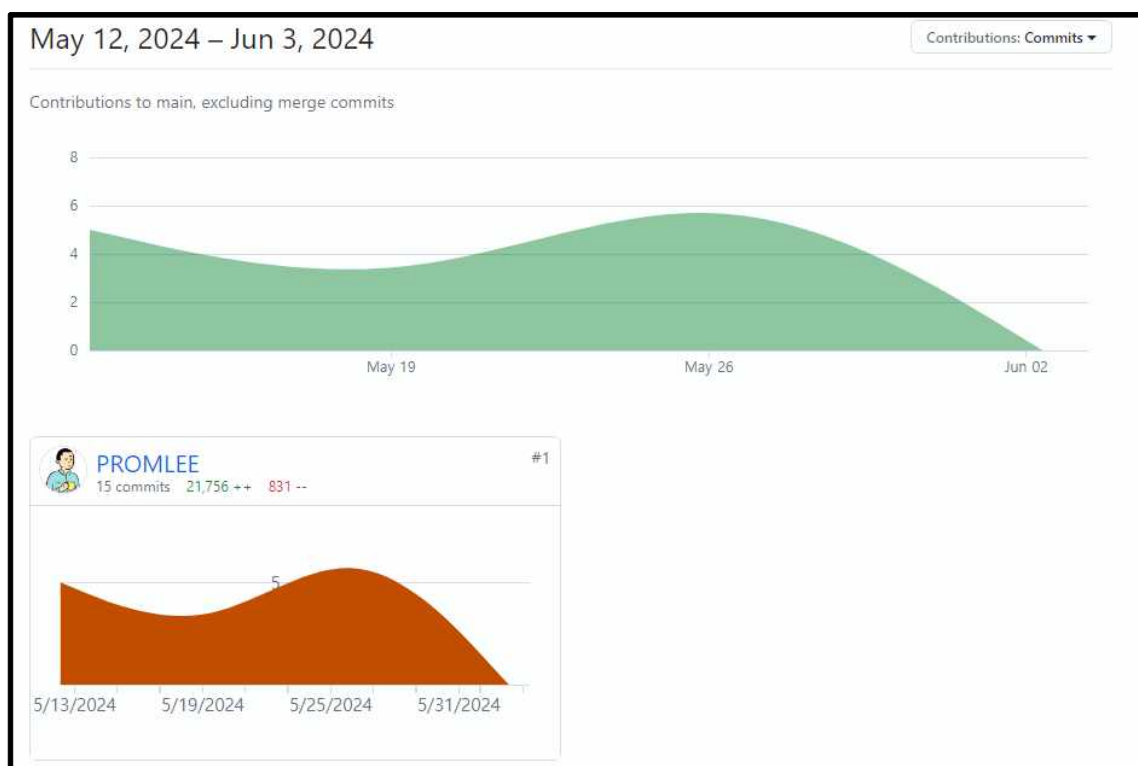
- ☑ 한동희 / 20210189 / Back-end
- ☑ 민보욱 / 20215744 / Back-end
- ☑ 전은진 / 20210117 / Front-end
- ☑ 이동훈 / 20216018 / Front-end
- ☑ 박상훈 / 20216942 / project 문서화



<Back-end contributions>



<GUI contributions>



<Web contributions>