

# Docking Design Specification

Team #4

2014311773 김용태

2014310444 김은하

2016314216 이상아

2009310261 임재현

위함흔

## *Table of Contents*

<b>1 Preface</b>	<b>7</b>
1.1 Objective	7
1.2 Readership	7
1.3 Document Structure	7
A. Preface	7
B. Introduction	7
C. System Architecture	7
D. Community System	7
E. User Information System	8
F. Matching System	8
G. Peristalsis System	8
H. Protocol Design	8
I. Database Design	8
J. Testing Plan	8
K. Development Environment	9
L. Development Plan	9
M. Index	9
1.4 Version of the Document	9
A. Version Format	9
B. Version Management Policy	9
C. Version Update History	9
<b>2 Introduction</b>	<b>10</b>
2.1 OBJECTIVES	10
2.2 Applied Diagrams	10
A. UML	10
B. PACKAGE DIAGRAM	11
C. DEPLOYMENT DIAGRAM	12
D. CLASS DIAGRAM	13
E. STATE DIAGRAM	13
F. SEQUENCE DIAGRAM	14
G. ER DIAGRAM	15
2.3 Applied Tools	15
A. Online Diagram Software	15
B. Front-End	15
C. Back-End	16
2.4 Project Scope	18
<b>3 System Architecture</b>	<b>21</b>
3.1 Objectives	21
3.2 System organization	21

A. Community System	22
B. User Information System	23
C. Matching System	24
D. Peristalsis System	25
3.3 context diagram	26
3.4 Usecase diagram	27
3.5 package diagram	28
3.6 deployment diagram	29
3.5 ER diagram	30
<b>4. Community System</b>	<b>31</b>
4.1 Objectives	31
4.2 Class Diagram	31
A. Account	31
B. Contents	32
C. Comments	32
D. Searching	33
4.3 Sequence Diagram	34
A. Community Main page	34
B. Uploading	35
C. Follow	35
D. Tag	35
E. Comments	36
F. Searching	37
4.4 State Diagram	38
A. Community Main page	38
B. Uploading	39
C. Follow	40
D. Tag	40
E. Comments	42
F. Searching	43
<b>5 User Information System</b>	<b>44</b>
5.1 Objectives	44
5.2 Class Diagram	44
A. Account	44
B. PetInfo	44
5.3 Sequence Diagram	45
A. Sign up with Naver	45
B. Sign up with Docking	46
C. Log in with Naver	46
D. Log in with Docking	46
E. Dog List Management	47

F. User Account Management	47
5.4 State Diagram	48
A. Sign in	48
B. Sign out	49
C. Log in	50
D. User account Access	51
<b>6 . Matching System</b>	<b>52</b>
6.1 Objective	52
6.2 Class Diagram	52
A. DB Handler	52
B. Matching	52
C. Direct Message	53
D. Walking Info	53
6.3 Sequence Diagram	54
6.4 State Diagram	55
<b>7 Peristalsis System</b>	<b>57</b>
7.1 Objective	57
7.2 Band Sync Sequence Diagram	57
7.3 Band working State diagram	58
7.4 Class Diagram	58
<b>8 Protocol Design</b>	<b>60</b>
8.1 Objectives	60
8.2 JSON	60
8.3 Protocol Description	60
A. Overview	60
B. Sign up with Naver Account Protocol	61
C. Sign up with Docking Account Protocol	61
D. Log in with Naver Account Protocol	61
E. Log in with Docking Account Protocol	62
F. Main Page Display Protocol	62
G. Contents Upload / Edit Protocol	62
H. Contents (include comments) Display Protocol	63
I. Comments Upload / Edit Protocol	63
J. Follow Protocol	63
K. Contents Search Protocol	64
L. Walking Information Save Protocol	64
M. Walking History Display Protocol	64
N. Walking Mate Recommend Protocol	65
O. Start Direct Message Protocol	65
P. Smartband Pairing Protocol	66

<b>9 Database Design</b>	<b>67</b>
9.1 Objectives	67
9.2 Entity-Relationship Model	67
9.2.1 Entities	70
A. User	70
B. Dog	71
C. Walking	71
D. Contents	71
E. Comments	72
9.2.2 Relationships	72
A. Tag; Contents	72
B. Follow	73
C. Raise	73
D. Take	74
E. Write; Contents	75
F. Have; Reply	75
G. Write; Comments	76
H. Have; Re-reply	77
I. Tag; Comments	77
9.3 Relational Schema	78
A. User	78
B. Dog	78
C. Walking	79
D. Contents	79
E. Comments	80
F. Tag	80
G. Follow	81
9.4 SQL DDL	81
A. User	81
B. Dog	82
C. Walking	82
D. Contents	83
E. Comments	83
F. Tag	83
G. Follow	84
<b>10 Testing plan</b>	<b>84</b>
10.1 Objective	84
10.2 Testing Policy	84
10.3 Test case	84
<b>11 Development Environment</b>	<b>89</b>
11.1 Objectives	89

11.2 Programming Language & IDE	89
11.3 Version Management Tool	90
11.4 Coding Rule	90
<b>12 Development Plan</b>	<b>92</b>
12.1 Objective	92
12.2 Gantt Chart	92
<b>13 Index</b>	<b>93</b>
13.1 Table Index	93
13.2 Figure Index	93
13.3 Diagram Index	94

# 1 Preface

## 1.1 Objective

Preface 에서는 이 문서의 대상 독자와 문서의 전체 구조, 문서 각 부분의 목적과 개요에 대해 서술한다. 또한 Version History를 제시하여 버전 관리 정책, 버전 변경 기록, 문서의 변경 사항들을 기술한다.

## 1.2 Readership

본 설계 명세서에서는 독자를 ‘DOCKING’을 개발하는 소프트웨어 엔지니어, 시스템 구조를 설계하는 아키텍처, 시스템을 유지 및 보수하는 소프트웨어 엔지니어, 그리고 개발에 참여하게 되는 모든 stakeholder들로 정의한다. 즉 본 문서의 독자는 본 문서에서 소개하는 시스템의 개발 및 유지 보수에 관련된 모든 구성원이다. 본 문서의 글자 형식은 나눔고딕이며, 가독성을 위해 표 등을 제외한 본문은 양쪽정렬하였다.

## 1.3 Document Structure

### A. Preface

Preface에서는 본 문서의 대상 독자들을 제시하고, 문서의 전체 구조와 각 장의 목적과 개요를 소개한다. 또한 버전 관리 정책과 이에 따른 Version History를 표 형태로 기록하고, 추가적인 문서의 변경사항을 서술한다.

### B. Introduction

Introduction에서는 시스템의 설계에 사용한 UML(Unified Modeling Language)의 다양한 다이어그램 작성을 위해 사용한 개발 툴을 소개하고, 이 프로젝트에서 개발 중인 ‘DOCKING’ 시스템의 scope에 대하여 설명한다.

### C. System Architecture

System Architecture에서는 목표 시스템에 대한 높은 수준의 개요와 시스템 기능의 전체적 분포를 보여준다. 전체 시스템의 구조는 Block Diagram을 통하여 도식화 하였으며, 서브 시스템들의 관계와 실제 배포 형태를 Package Diagram 및 Deployment Diagram을 사용하여 표현하였다.

### D. Community System

Community System은 유저들이 사용하는 커뮤니티와 관련된 시스템으로, 반려인, 비반려인, 전문가 모두가 이 시스템에 관여한다. Community System은 Follow sub-system, Upload sub-system, Search System과 같은 3가지 sub-system으로

나뉜다. Class diagram, Sequence diagram, State diagram을 통해 Community System의 구조를 표현하고 설명한다.

## E. User Information System

User Information System은 사용자 정보와 관련된 시스템으로, 사용자의 계정을 생성하고, 확인하고, 로그인하고, 계정에서 올려진 게시글에 접근하기 위한 서브시스템을 제공한다. User Information System은 User account create/deleter sub-system, User account verifier sub-system, User account access sub-system, user post access module과 같은 서브시스템들과 모듈로 구성된다. Class diagram, Sequence diagram, State diagram을 통해 User와 Information System의 상호작용 및 시스템 구조를 설명한다.

## F. Matching System

Matching System은 산책 메이트 매칭과 관련된 시스템으로, 실제 산책 모임이 진행되기에 앞서 각 반려인 유저들에게 적합한 산책 메이트를 추천해주는 역할을 한다. Matching System은 Walking Mate Recommendation sub-system, Direct Message sub-system으로 구성된다. Class diagram, Sequence diagram, State diagram을 통해 User들을 어떤 방식으로 매칭시켜주는지 시스템 구조를 설명한다.

## G. Peristalsis System

Peristalsis System은 반려견이 착용할 스마트 밴드와 관련된 시스템으로, 반려견의 스마트 밴드와 반려인의 스마트폰을 연동할 수 있도록 해준다.

## H. Protocol Design

Protocol Design에서는 서브시스템들이 상호 통신하기 위하여 필수적으로 준수해야 하는 프로토콜에 대하여 서술한다. 통신하는 메시지의 형식과 용도, 의미를 설명한다.

## I. Database Design

Database Design에서는 요구사항 명세서에서 기술하였던 요구사항을 기반으로 한 데이터베이스를 설계하고 이에 대하여 설명한다. 요구사항에 기반한 데이터베이스의 ER Diagram을 작성한 뒤 이를 기반으로 Relation model 또한 작성한다. 그 이후 Normalization 과정을 통하여 데이터베이스 사이에 존재할 수 있는 중복을 방지하고, SQL DDL을 작성한다.

## J. Testing Plan

Testing Plan에서는 요구사항 명세서에서의 관리자 및 사용자 시나리오를 바탕으로 한 전체 시스템의 테스트 케이스를 설명하고, 관련 테스트를 수행하기 위한 테스트 정책을 제시한다.



## K. Development Environment

Development Environment에서는 실제 시스템 개발을 위하여 필요한 개발 환경과 코딩 규칙을 설명한다. 개발 과정에서의 버전 관리 도구를 제시하고, 프로그래밍 과정에서 기반이 되는 규칙들에 대하여 서술한다.

## L. Development Plan

Development Plan에서는 이후 진행할 개발 계획에 대해 서술한다.

## M. Index

Index에서는 문서의 인덱스들을 정리한다. 알파벳 순서의 단어 인덱스를 기반으로, 다이어그램 인덱스 및 기능 인덱스를 포함한다.

# 1.4 Version of the Document

## A. Version Format

버전 번호는 major.minor[.maintenance] 형태로 표현하며 본 문서의 버전은 0.1부터 시작한다.

## B. Version Management Policy

본 설계 명세서가 수정될 때마다 버전 넘버를 업데이트한다. 이미 완성되어 정리된 부분에 대한 변경의 경우는 minor number를 업데이트하고, 새로운 부분을 추가하거나 문서의 구성에 괄목한 변화가 있을 경우 major number를 업데이트 한다. 이미 작성한 부분에 대해 오류를 수정하는 경우 maintenance number를 추가하거나 변경한다.

## C. Version Update History

version	date	explanation
0.1	2019.05.15	Preface, Introduction, System Architecture 작성
0.5	2019.05.17	Sub Program에 대한 상세 Design 작성
0.8	2019.05.18	Development Environment 작성
1.0	2019.05.19	Protocol Design, Database Design, Testing Plan, Index 작성, 양식 통일, 최종수정

## 2 Introduction

### 2.1 OBJECTIVES

Introduction 에서는 본 문서에서 시스템을 설계할 때 사용하는 모든 종류의 다이어그램 및 툴에 관해 소개하고 설명한다.

### 2.2 Applied Diagrams

#### A. UML



<Figure 2.1 UML>

통합 모델링 언어 (Unified Modeling Language, 이하 UML) 는 객체 지향 소프트웨어 설계를 위해 사용되던 여러 종류의 다이어그램들을 통합하여 만든 모델링 표기법으로, 시스템 개발 과정에서 개발자간의 의사소통이 원활하게 이루어지게 하기 위한 객체 지향적 분석과 설계 방법론의 표준 지정을 목표로 하고 있다. 1994년 소프트웨어 방법론의 선구자인 Grady Booch, James Rumbaugh, Ivar Jacobson 에 의해서 연구되었고, 1997년 객체 관리 그룹 (OMG, Object Management Group) 에서 객체 모델링 기술 (OMT; object modeling technique), OOSE 방법론 등을 통합하여 UML을 발표하였다.

UML은 요구 분석, 시스템 설계, 시스템 구현 등의 과정에서 생길 수 있는 개발자 간의 의사소통 불일치를 해소할 수 있다는 장점 덕분에, 현재 객체 지향 시스템 개발 분야에서 가장 우수한 모델링 언어로 인식되고 있다. UML은 개발자가 구축 하고자 하는 소프트웨어 시스템을 구현하기에 앞서서 표준화되고 이해가 쉬운 방식으로 설계되어, 소프트웨어에 관련된 모든 사람들과의 상호 소통을 가능하게 하는 효율적인 매개체 역할을 한다. 이때문에 생략되거나 불일치한 모델링 구조에 대한 지적도 용이하고, 그 스스로 모델링에 대한 표현력이 강하고 비교적 모순이 적은 논리적인 표기법 (notation)을 가진 언어라는 장점이 있다.

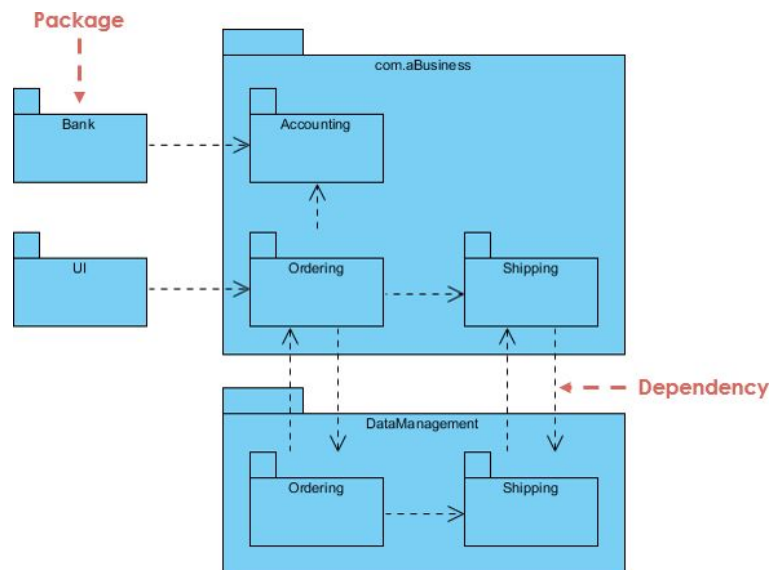
UML 이전의 시스템 개발에서 구현에 가장 큰 영향을 미치는 요인은, 시스템 의뢰인과 개발자 사이의 의사소통이 아닌 개발자들의 자의적인 이해와 논리였다. 그 결과로

개발된 시스템이 의뢰인의 입장에서 만족스럽지 않거나 의뢰인의 의도에 부합하지 않는 시스템이 개발되는 것은 필연적인 일 이었다.

이러한 문제를 보완하기 위해서는, 시스템 구축 이전에 의뢰인과 개발자 사이의 충분한 의사소통과, 이를 통해 개발하려는 시스템의 기능과 동작을 확실하고 철저하게 계획하는 과정이 필수적이다. 현재의 시스템 개발에는 수행하는 일이 다른 많은 팀이 필요하고, 하나의 팀은 다시 여러 개인으로 구성된다. 팀의 각 멤버는 자신의 역할을 충분히 파악해야 하고, 그 역할을 막론한 모든 사람들이 이해하고 동의할 수 있는 방법으로 설계 과정을 조직화해야 한다. 수행 업무가 다른 여러 팀들 사이에서의 충분한 소통 및 시스템 설계 과정의 표준화 또한 완전한 시스템 개발을 위해 필요로 되는 사항이다. 이에 더해, 의뢰인은 개발 팀이 어떤 일을 해야 하는지 이해하고, 개발팀이 의뢰인의 요구를 제대로 이해하지 못했을 경우에는 변경 사항을 지적해 줄 수 있어야 한다. 시스템 개발에서 분석가, 의뢰인, 개발자가 서로 오해없이 이해할 수 있고, 시스템 설계를 일반적으로 표현할 수 있는 수단에 대한 필요성이 중요해진 것이다. UML은 바로 이러한 표준화 수단을 제공하기 위해 마련한 표기법이다.

UML은 유스케이스(usecase) 다이어그램, 클래스 다이어그램 등 8개의 다이어그램을 기반으로 객체 지향 소프트웨어를 개발하기 위한 풍부한 분석 및 설계 장치를 제공하고 있기 때문에 향후 상당기간 동안 산업계의 표준으로 활용될 것이라 예상된다. 또한, 구축할 시스템의 유형에 관계 없이 모든 시스템에 대해 적용될 수 있고, 시스템 개발 방법론, 개발에 사용할 프로그래밍 언어, CASE도구에 관계 없이 적용될 수 있는 일반적인 표현 방법이기 때문에 UML을 이용하여 시스템 설계를 하였다.

## B. PACKAGE DIAGRAM



<Figure 2.2 package diagram example>

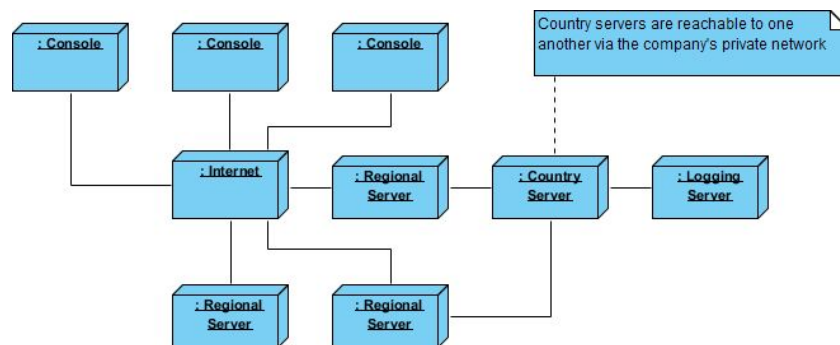
문제 도메인에 대해 더 깊이 이해할수록 개념적 모델이 더 복잡해지고 관리할 수 없을 정도로 크기가 커질 수 있다. 이렇게 복잡해진 시스템을 이해하기 위해서는 추상적인 개념들을 하나의 그룹으로 묶어서 용이하게 할 수 있다. 이렇게 만든 그룹들은 각각

클래스와 같은 개념으로 많은 인스턴스들을 가지고, 오로지 시스템을 이해하기 위한 목적으로만 존재하게 된다. 이러한 그룹을 패키지(Package)라고 하는데, 패키지를 사용하면 전체 모델을 크기가 보다 작고 관리하기 쉬운 하위 집합으로 나눌 수 있다.

패키지는 UML 시스템 모델의 기본 구성 요소로, 요소들을 그룹으로 조직하기 위한 범용 메커니즘으로써 시스템모델의 요소들을 조직하고 이해할 수 있도록 해준다. 패키지 안에 담기는것은 클래스에만 국한되지 않고, 하위 패키지들과 Use Case, Activity Diagram 등도 담을 수 있으며, 패키지의 표시 여부는 물론 패키지가 포함하는 요소의 표시 여부를 설정할 수 있다. 패키지를 구성할 때에는 여러 사람이 동의할 수 있는 형태로 구성되어야 하며, 패키지의 구성과 이름 체계는 개발자들이 쉽게 이해하고 사용할 수 있어야 한다.

패키지 내부의 모든 클래스들은 다양한 기능적 측면에서 유사한 면을 가진다. 패키지 내부의 클래스들은 서로 밀접한 관련성을 가지며, 다른 패키지의 클래스들과는 의존관계가 약하다. 이와 같이 패키지 다이어그램은 패키지 삽입 및 패키지 확장을 포함하여 모델 요소들을 패키지로 그룹핑함으로써 조직 및 요소의 독립성을 묘사하고, 요소들 간의 관계를 시각화하여 제공한다.

## C. DEPLOYMENT DIAGRAM

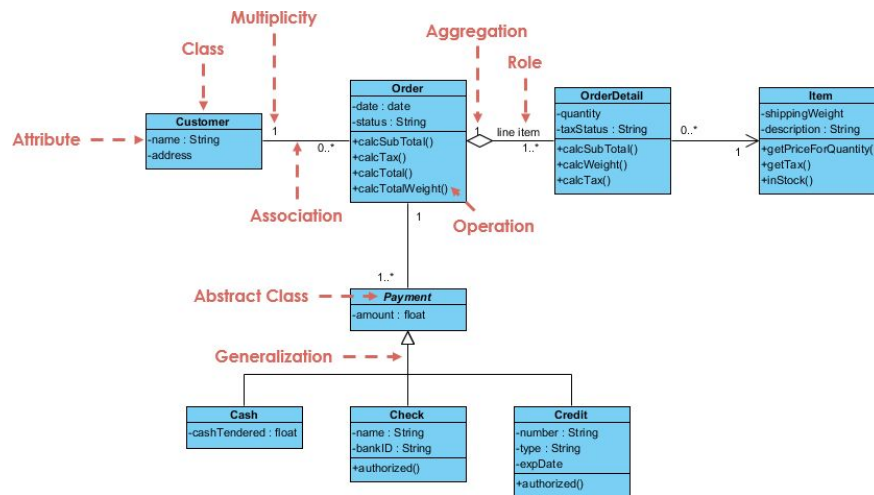


<Figure 2.3 deployment diagram example>

배치 다이어그램 (DeploymentDiagram) 은 네트워크, 하드웨어 또는 소프트웨어들을 실행 파일 수준의 컴포넌트들과 함께 표현한 다이어그램이다. 이들은 시스템을 구성하는 하드웨어간의 연결 관계를 표현하고, 하드웨어 자원에 대한 소프트웨어 컴포넌트의 배치 상태를 표현한다. 즉, 시스템이 실행되는 환경인 노드와 그 노드에 배치된 컴포넌트의 구성, 각 노드들 사이의 네트워크 특성이나 프로토콜과 같은 관계를 나타내는 다이어그램이다. 노드는 처리 능력을 가진 장치를 의미하며, 각각의 노드에는 프로세서나 디바이스들에 대한 사항을 표현하고 Deployment Diagram 에서 직육면체로 표기한다.

Deployment Diagram 은 시스템 설계 단계의 마지막에 작성되는데, 이는 모든 설계가 마무리 되어야 소프트웨어의 컴포넌트가 정의되고, 시스템의 하드웨어 사양도 확정 되기 때문이다. 배치 다이어그램은 소프트웨어 시스템이 배치, 실행될 하드웨어의 자원들을 정의하고, 소프트웨어의 컴포넌트가 어떤 하드웨어 자원에 탑재되어 실행될지를 정의하는 목적을 가지고 있다. Deployment Diagram 은 운영 환경이 중요한 요소로 평가되는 개발과정에서는 많은 Use Case 기술서나 클래스 다이어그램들보다도 그 의미가 더 중요한 문서가 될 수도 있다.

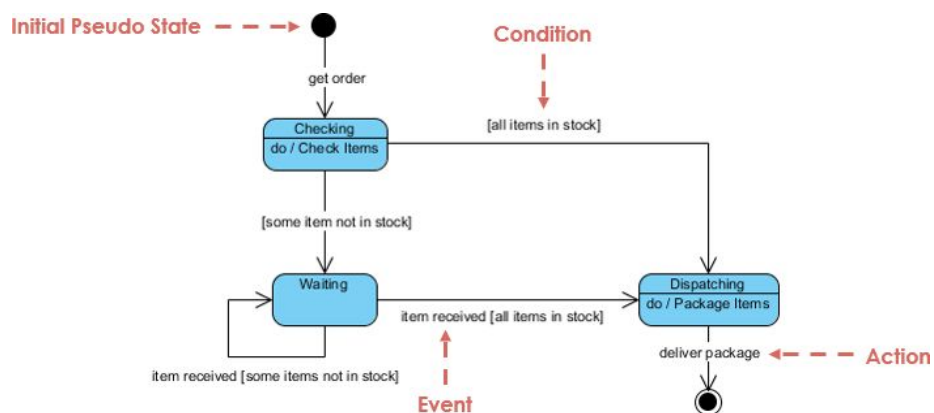
## D. CLASS DIAGRAM



<Figure 2.4 class diagram example>

Class Diagram은 객체 지향 모델링에서 가장 널리 사용되는 다이어그램으로, 시스템의 정적인 상태의 논리적인 구조를 표현하는 다이어그램이다. 클래스(Class)란 객체 지향 프로그래밍에서 속성(Attribute)과 행위(메소드(Method), 또는 멤버 함수(Member Function))를 갖는 하나의 객체 단위이다. 시스템의 클래스와 클래스 Attribute, 클래스들의 관계를 나타낼 수 있기 때문에 객체 지향 시스템의 상속 등의 관계를 명확하게 표현할 수 있다. Class Diagram은 객체 지향 다이어그램에서 가장 중요한 요소이며, 코드 생성의 직접적인 원인이 되기 때문에 프로그래밍 개념과 같은 의미의 표현(Notation)을 통해 도식화한다.

## E. STATE DIAGRAM

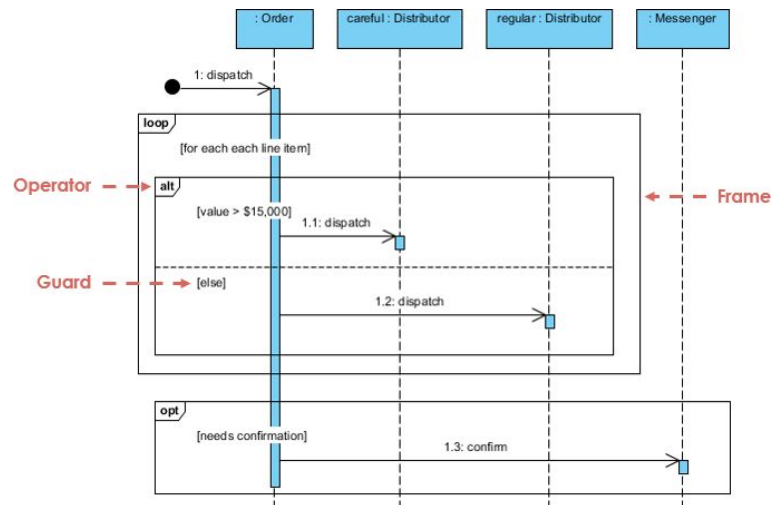


<Figure 2.5 state diagram example>

State Diagram은 객체 지향 모델링에서 클래스의 인스턴스 사건(Event)에 의거한 시스템의 전체적인 작동을 상세히 기술하는 다이어그램이다. State Diagram은 상태의 변화에 의한 동작 또는 하나의 상태에서 다른 상태로 변화되게 하는 사건의 주어진 시간 동안의 상태를 나타낸다. 다시 말해, State Diagram은 상태와 상태의 변화를 포함하는데, 이러한 표기는 실제 구현에서 UI를 정의하는데 도움을 준다.

상태 다이어그램은 어떤 이벤트에 대한 반응적인 (Reactive) 특성을 가지는 객체에 대해 기술하기 위해 이용되며, 객체가 갖는 여러가지 상태를 표현한다. 객체는 기존 상태에 따라 동일한 메시지에 대해서 다른 행동을 보이기 때문에, 상태 다이어그램을 통해 시스템 내의 객체가 가질 수 있는 상태가 어떤 것이 있는지에 대해, 또 각 상태를 나타낼 때 특정 이벤트에 대해 어떤 반응을 보일지에 대해 기술한다.

## F. SEQUENCE DIAGRAM



<Figure 2.6 sequence diagram example >

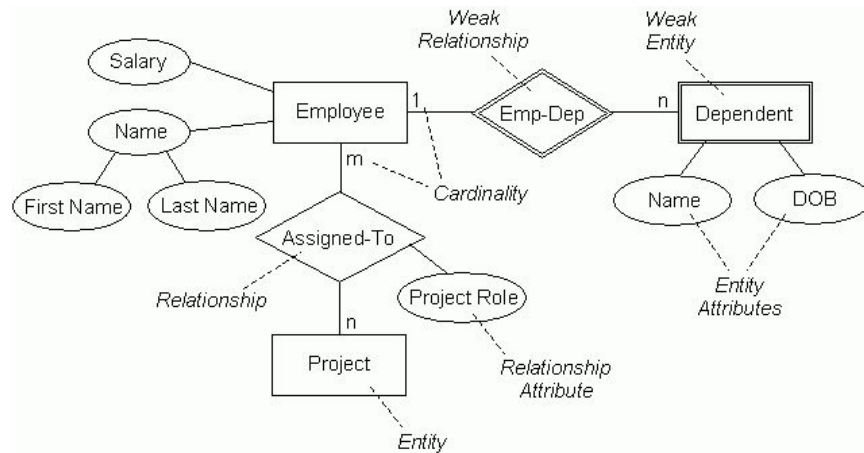
Sequence Diagram은 시스템 내에서의 각 컴포넌트들이 주고 받는 메시지의 흐름을 시간 순차적으로 표현하는 상호작용 다이어그램이다. 상호작용 다이어그램은 다이어그램의 수직 방향이 시간을 나타내고, 메시지 교류의 주체인 객체와, 객체를 통해 실질적인 데이터를 주고 받는 메시지(오퍼레이터)를 보여주는 역할을 하며, 그 구성 요소로는 Actor, 객체, 메시지, 회귀 메시지, 제어 블록 등이 있다.

Sequence Diagram은 각 컴포넌트들의 상호 작용이 명확히 보이기 때문에 각 컴포넌트간의 관계와 각 컴포넌트들이 갖고 있는 속성, 행동들을 더욱 명확히 할 수 있다. Sequence Diagram은 시스템 내부의 동적인 움직임을 표현해 주는 다이어그램이기 때문에, 속성 및 함수로 이루어진 Class Diagram을 동적 프로그래밍으로 설계하는 중요한 과정이다.

Sequence Diagram은 Use-Case Diagram과 자주 같이 사용되는데, 이는 각 Use-Case를 상세히 표현하는데 Sequence Diagram을 사용하는 것이 유리하기 때문이다. Sequence Diagram은 Use-Case를 실현한다. Use-Case Diagram에서는 시스템이 제공해야 하는 서비스를 정의하기 때문에, Use-Case는 프로그램으로 구현되기 전에 Sequence Diagram으로 설계 되어야 한다. 따라서 Sequence Diagram은 각 Use-Case 별로 작성되며 Use-Case에 필요한 객체가 주인공으로 등장하고, 객체 간 메시지를 통해서 Use-Case의 기능이 실현된다.

Sequence Diagram에서는 객체의 오퍼레이션과 속성을 상세히 정의한다. 이는 객체간 상호작용을 정의하는 과정에서 객체들이 가져야 하는 오퍼레이션과 속성을 구체적으로 정의할 필요가 있기 때문이다. 객체는 다른 객체가 의뢰하는 일을 처리하는 객체의 책임으로서, 객체의 책임은 오퍼레이션으로 정의되어야 하며, 이 행위를 위해 필요한 객체의 속성도 정의되어야 한다.

## G. ER DIAGRAM



<Figure 2.7 ER diagram example>

ER Diagram은 데이터 베이스에서 구조화된 각 데이터 개체들의 관계를 표현하기 위해 사용하는 다이어그램으로, UML에 포함되지는 않는다. 데이터가 저장되는 공간인 데이터 베이스의 구조 및 그에 수반한 제약 조건들은 다양한 기법에 의해 정의될 수 있는데, 그 기법 중 하나가 개체-관계모델링 (Entity-Relationship Modelling)이다. 이 ER 모델링 프로세스의 산출물이 ER Diagram (Entity-Relationship Diagram) 인데, 여기에서 개체(Entity)란 현실 세계의 객체로서 유형 또는 무형의 정보를 대상으로 서로 구별할 수 있는 것이며, 관계(Relationship)란 두 개 이상의 개체 사이에 연관성을 기술한 것이다. 따라서, ER Diagram은 조직의 정보자원(Resource)을 전반적으로 계획하는데 있어서 필수적이며 유용한 도구이다.

## 2.3 Applied Tools

### A. Online Diagram Software

#### (1) draw.io

명세서에 포함되는 다양한 다이어그램 작성을 위해 draw.io를 활용한다.



<Figure 2.8 drawio>

### B. Front-End

#### (1) Android Studio & Android API



안드로이드 앱 개발을 위해 Android Studio & Android API를 활용한다. 버전은 3.4를 사용한다.



<Figure 2.9 Android Studio>

(2) OpenGL ES

커뮤니티 업로드와 관련하여 사진찍기 및 카메라 기능을 제공하기 위해 OpenGL ES를 사용한다. Open GL ES는 임베디드 시스템을 위한 그래픽스 라이브러리이다.



<Figure 2.10 OpenGL ES>

## C. Back-End

(1) Firebase

Firebase는 모바일 및 웹 애플리케이션 개발 프로그램이다. Firebase를 통해 기존 DBMS를 사용하는 것보다 손쉽게 사용자 인증 기능 등을 구현할 수 있다.



<Figure 2.11 Firebase>

(2) AWS EC2

어플리케이션의 서버 구축을 위해 Amazon web Services를 활용한다. 인스턴스로는 EC2를 사용한다.





<Figure 2.12 Amazon web Services>

(3) Google Map API

산책 중 유저들의 위치 정보를 제공받기 위해 google map API를 사용한다.



<Figure 2.13 Google Map API>

(4) OpenWeatherMap API

산책 시 날씨 정보를 제공받기 위해 google map API를 사용한다.



<Figure 2.14 OpenWeatherMap API>

(5) Arduino RetroBand

RetroBand는 오픈소스로 제공되는 아두이노 웨어러블 디바이스로, 가속도 센서를 이용하여 수집한 데이터를 모바일 폰으로 보내주는 기능을 한다. Arduino RetroBand를 활용하여 반려견용 스마트밴드를 구현한다.



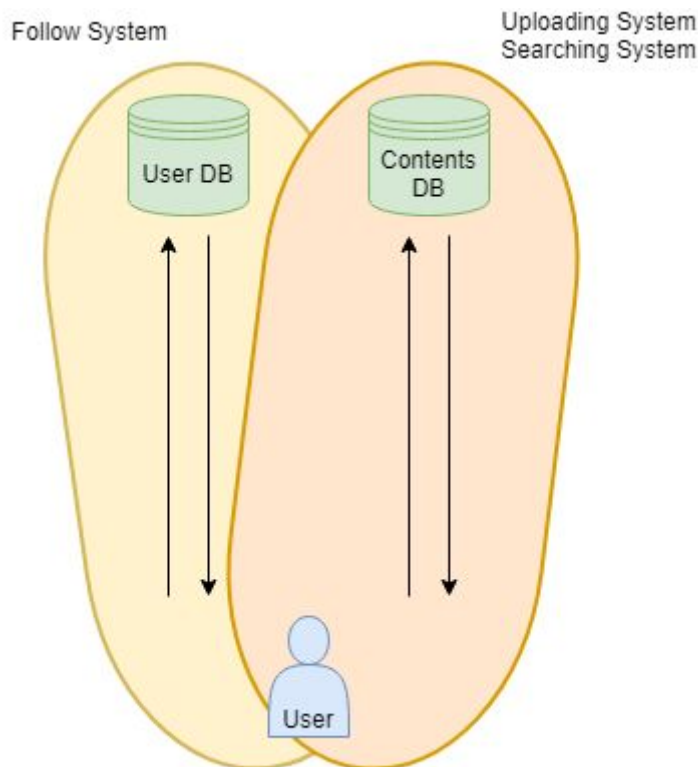
〈Figure 2.15 Arduino〉

## 2.4 Project Scope

Docking!은 인공지능을 활용해 산책 메이트를 매칭해줌으로써 반려견의 사회성 증진 및 반려인의 산책 동기부여를 높여주는 동시에 반려인들을 위한 커뮤니티를 만들어 정보 공유를 하게 해 주는 시스템이다. Docking!은 크게 커뮤니티 시스템, 유저 정보 시스템, 매칭 시스템, 연동 시스템이라는 네 개의 서브시스템으로 구성되어 있다.

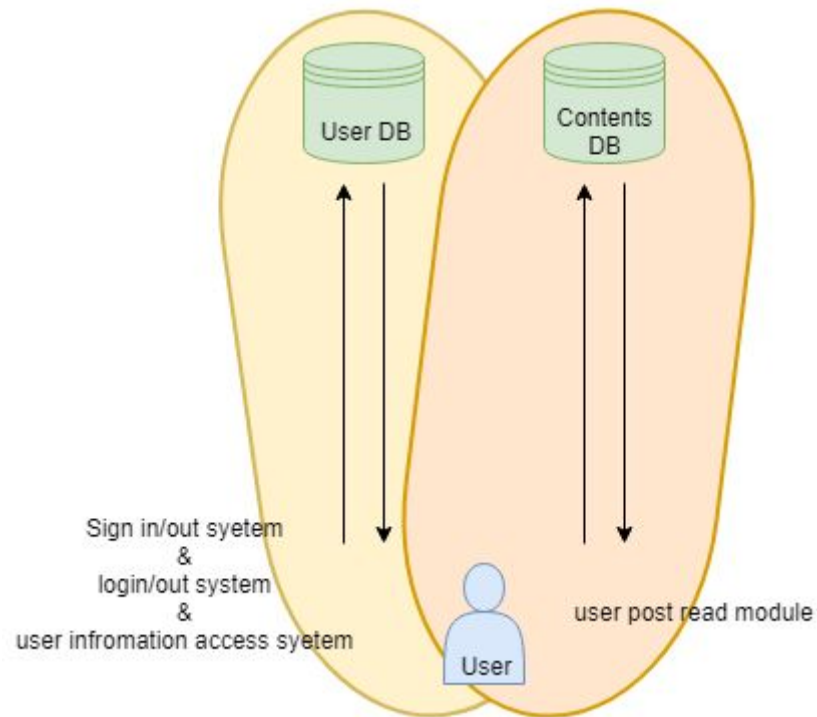
먼저 기존의 소셜 네트워킹 사이트에서 제공하는 기능을 기반으로 설계된 시스템으로 Community System이 있다.

Community System은 반려인들 간 정보공유 등과 관련된 시스템으로 하위 세 개의 시스템으로 분류된다. 계정 간 Following을 관리하는 Follow System과 contents uploading을 관리하는 Uploading System, 콘텐츠 및 게시글을 검색할 수 있는 Searching System으로 각 하위 시스템이 구성되어 있다. 해당 시스템의 구조는 다음과 같다.



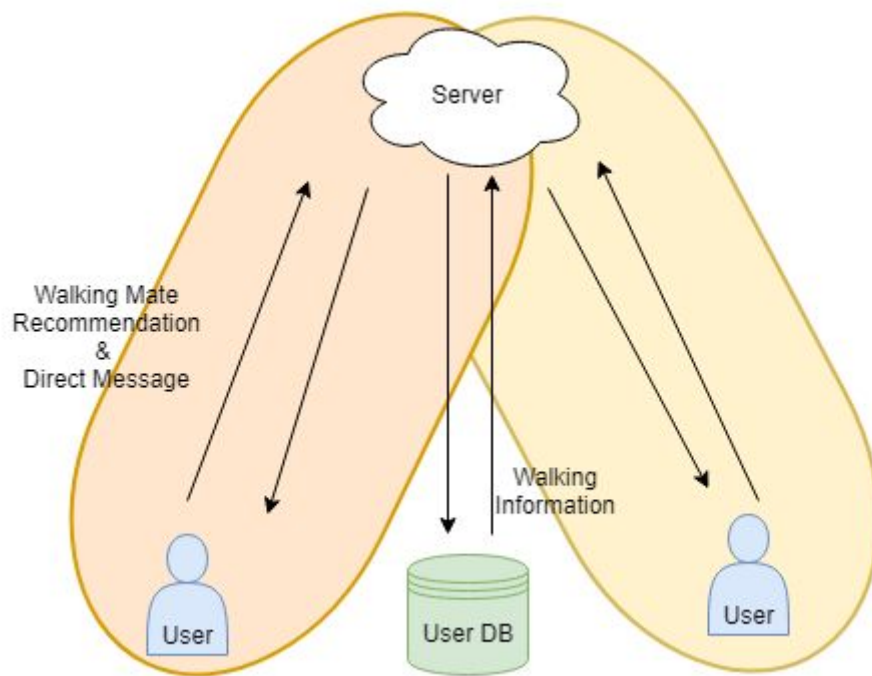
〈Diagram 2.1 Community System〉

User information System은 사용자 계정의 생성, 삭제를 관리하며 사용자 계정에의 접근을 제어해 Login/out 기능에 관여한다. 또한 사용자 정보의 접근을 관리하여 User 자신의 post를 조회할 수 있도록 한다.



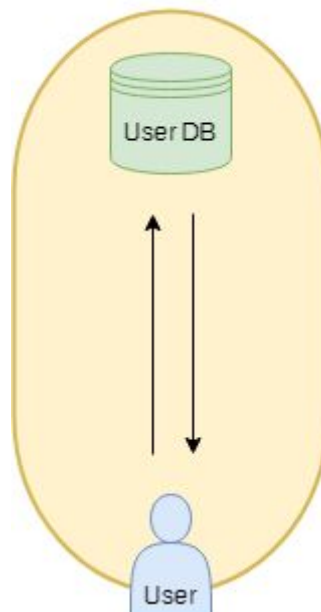
<Diagram 2.2 User information System>

Matching System은 산책 mate를 원하는 이용자끼리의 매칭을 도와주는 서비스이다. 기본적인 사용자의 정보와, DM서비스를 이용해 구동된다. Server는 유저들의 Database를 참고하여 적합한 mate와의 매칭이 이루어 질 수 있도록 돕는다.



<Diagram 2.3 Matching System>

마지막으로 반려견과의 산책에서 활용될 스마트 밴드와 반려인의 스마트폰을 연동하여 사용하기 위한 Peristalsis System이 있다.  
해당 시스템은 User Information system과 연계되며 유저 데이터베이스에 관여한다.



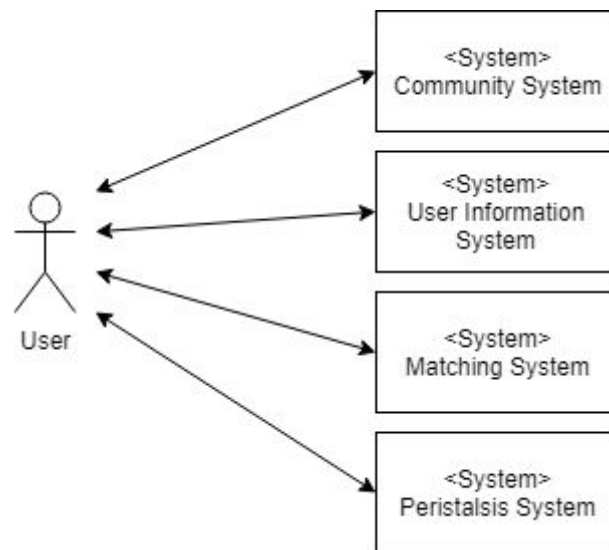
<Diagram 2.4 Peristalsis System>

## 3 System Architecture

### 3.1 Objectives

System Architecture에서는 현재 개발하는 시스템의 전체적인 구조에 대해 서술한다. 또한 Block Diagram, Package diagram과 Deployment diagram을 이용해 시스템이 실제로 어떻게 사용되는지 설명한다.

### 3.2 System organization

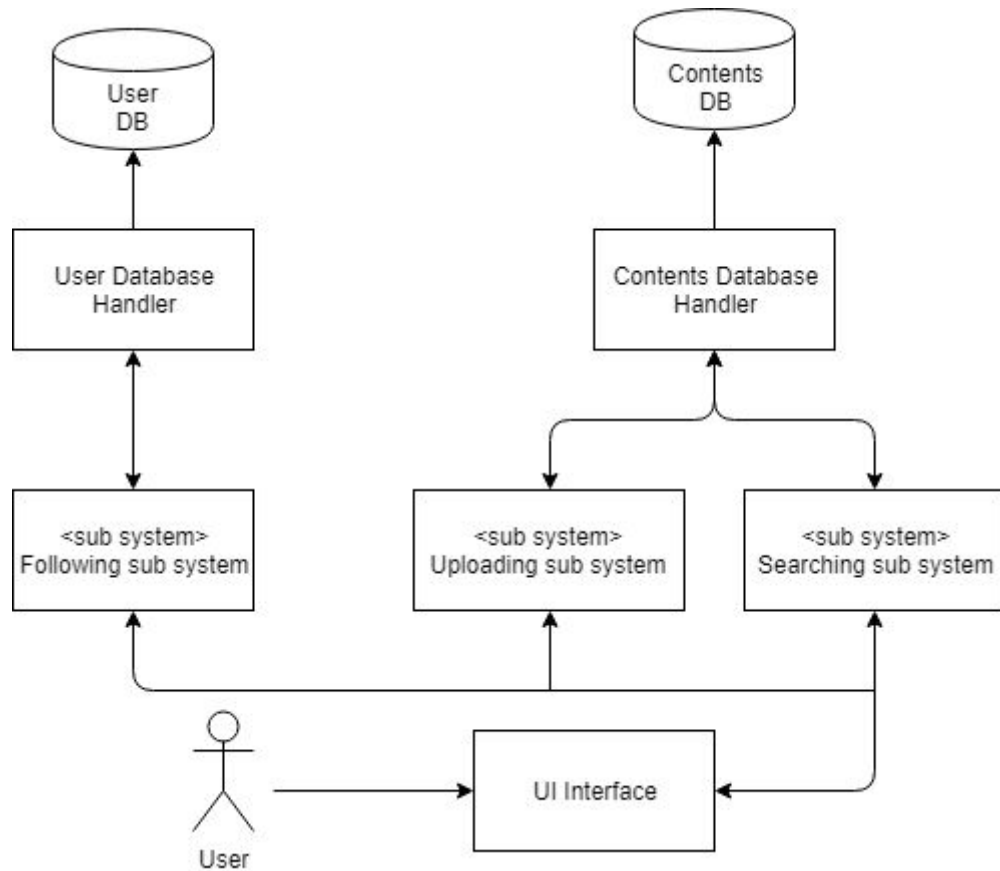


〈Diagram 3.1 System Organization Block Diagram〉

Docking!은 Android Studio를 이용하여 구현된다. 서버는 Community System, User Management System, Matching System, Peristalsis System 기능을 제공하기 위해 데이터베이스에 접근하여 입력, 출력, 수정을 하는 개체이다.

## A. Community System

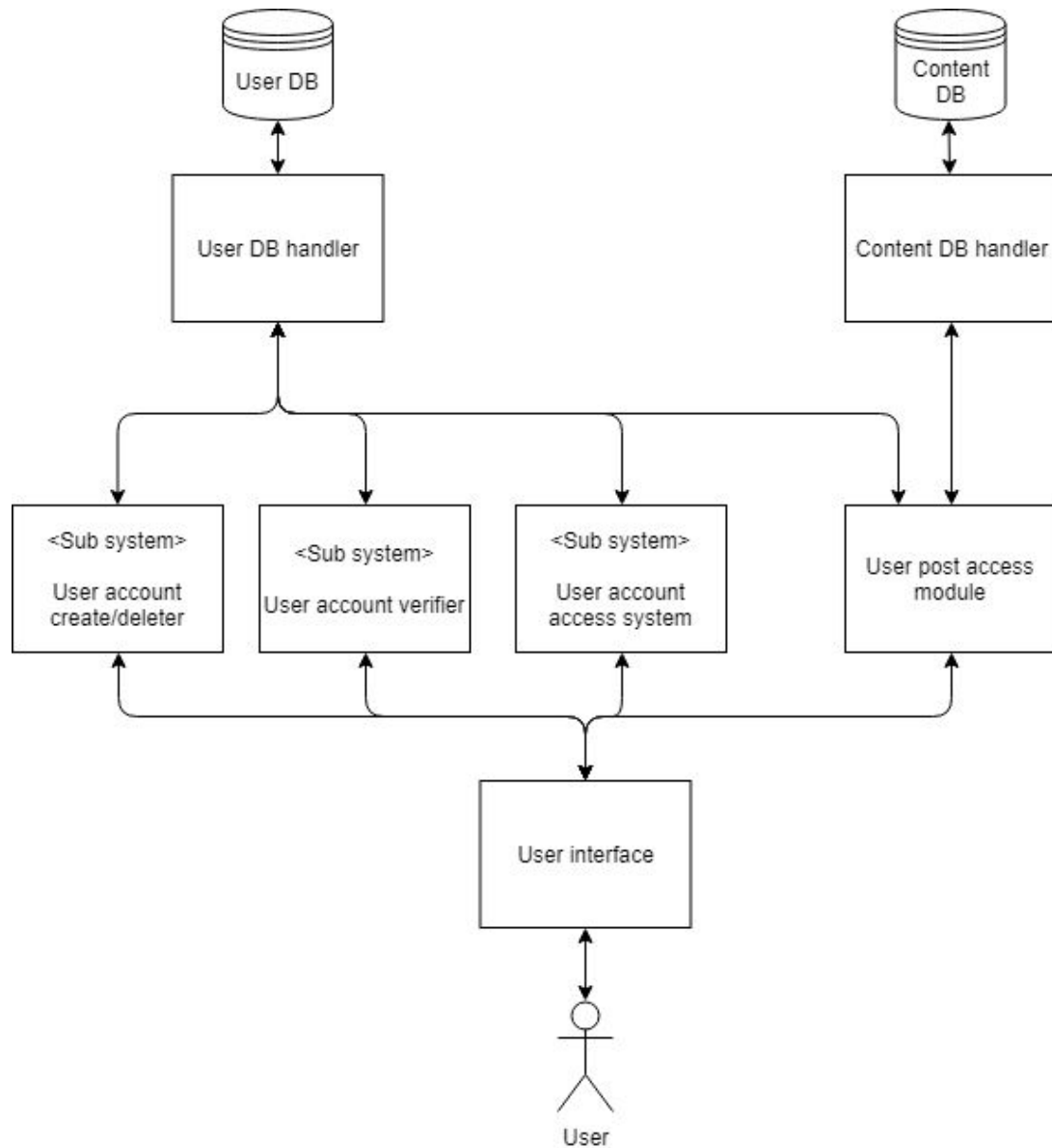
Community System은 반려인들 간 커뮤니티와 관련된 시스템이다. 사용자가 다른 사용자를 팔로우하는 시스템과 자신의 게시물을 업로드하는 시스템, 다른 사용자들의 게시글을 검색하는 3개의 하위 시스템으로 나뉜다.



<Diagram 3.2 Community System Architecture>

## B. User Information System

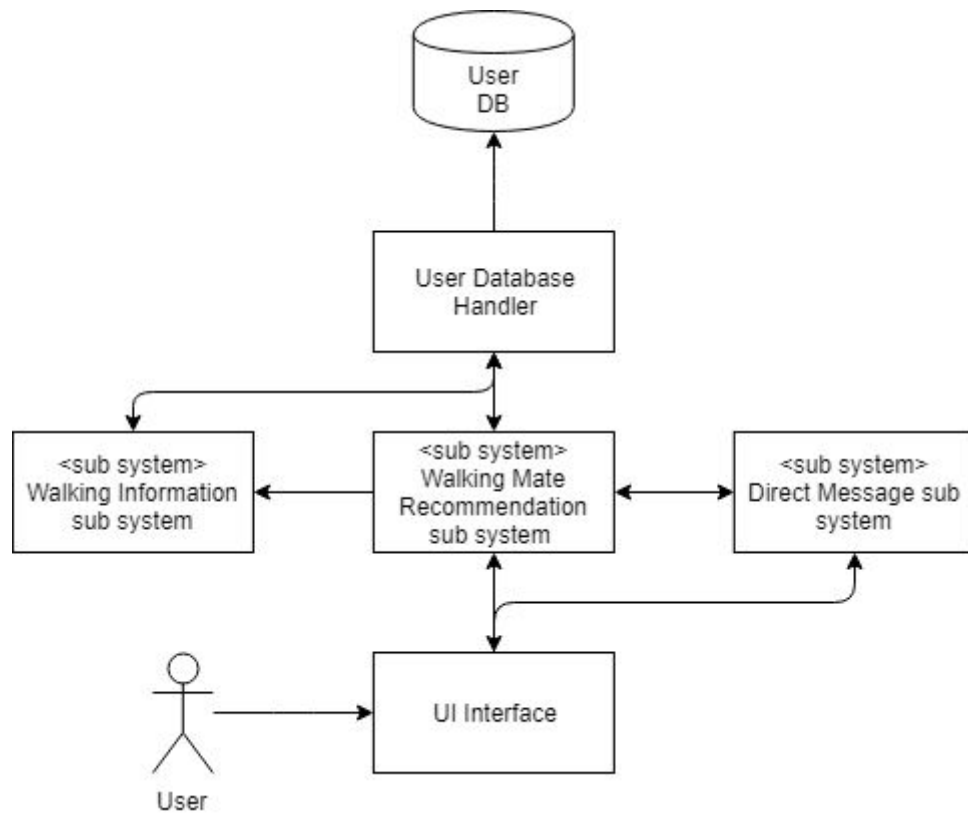
User Information System은 사용자의 정보를 생성,삭제하거나 접근하는데 사용되는 시스템이다.



<Diagram 3.3 User Information System Architecture>

### C. Matching System

Matching System은 산책 mate를 원하는 이용자가끼리의 매칭을 도와주는 서비스이다. 사용자와 다른 사용자 간의 산책mate 매칭, 산책 정보, 매칭된 유저와의 Direct Message를 제공하는 세 개의 서브시스템으로 구성된다.

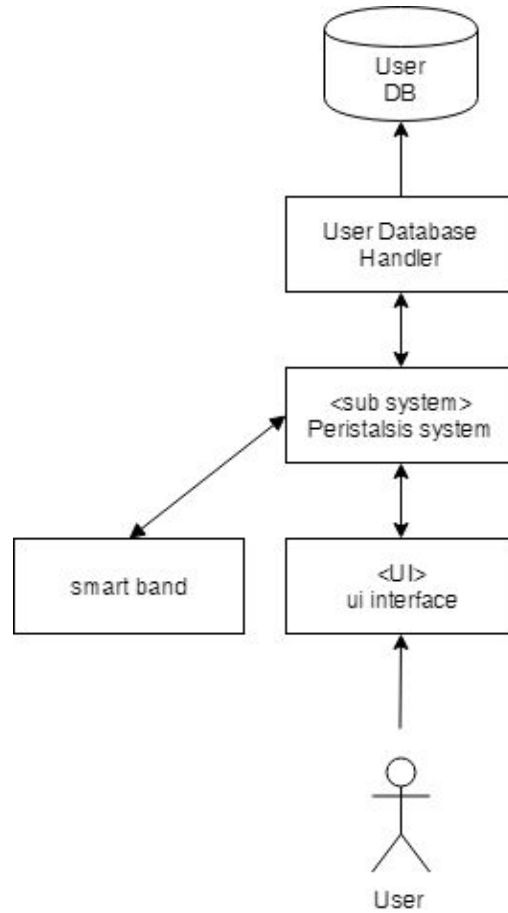


〈Diagram 3.4 Matching System Architecture〉



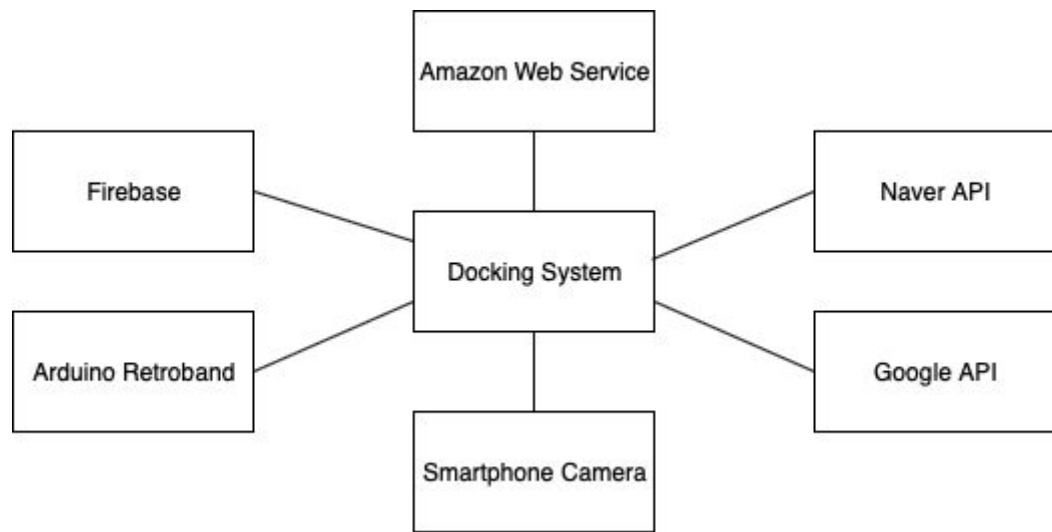
## D. Peristalsis System

Peristalsis System은 반려견이 착용할 스마트 밴드와 관련된 시스템이다. 반려견의 스마트 밴드와 반려인의 스마트폰을 연동하여 사용한다.



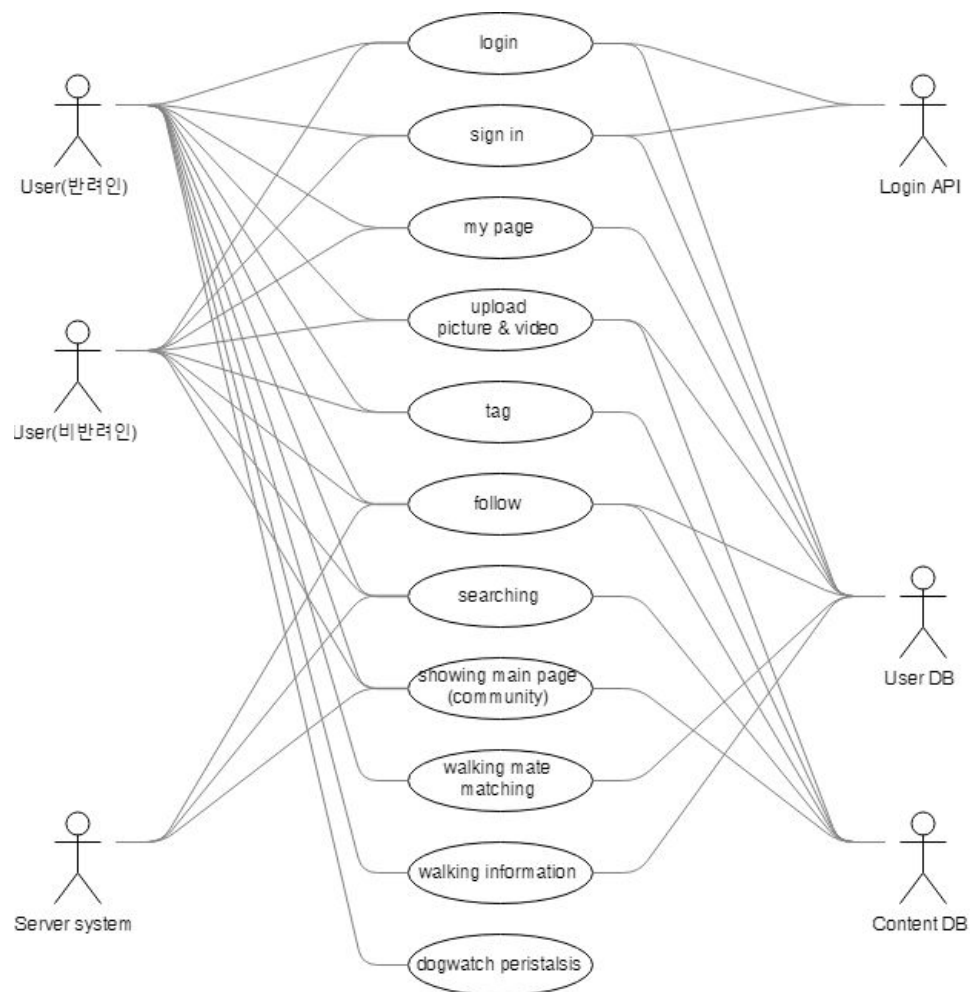
〈Diagram 3.5 Peristalsis System Architecture〉

### 3.3 context diagram



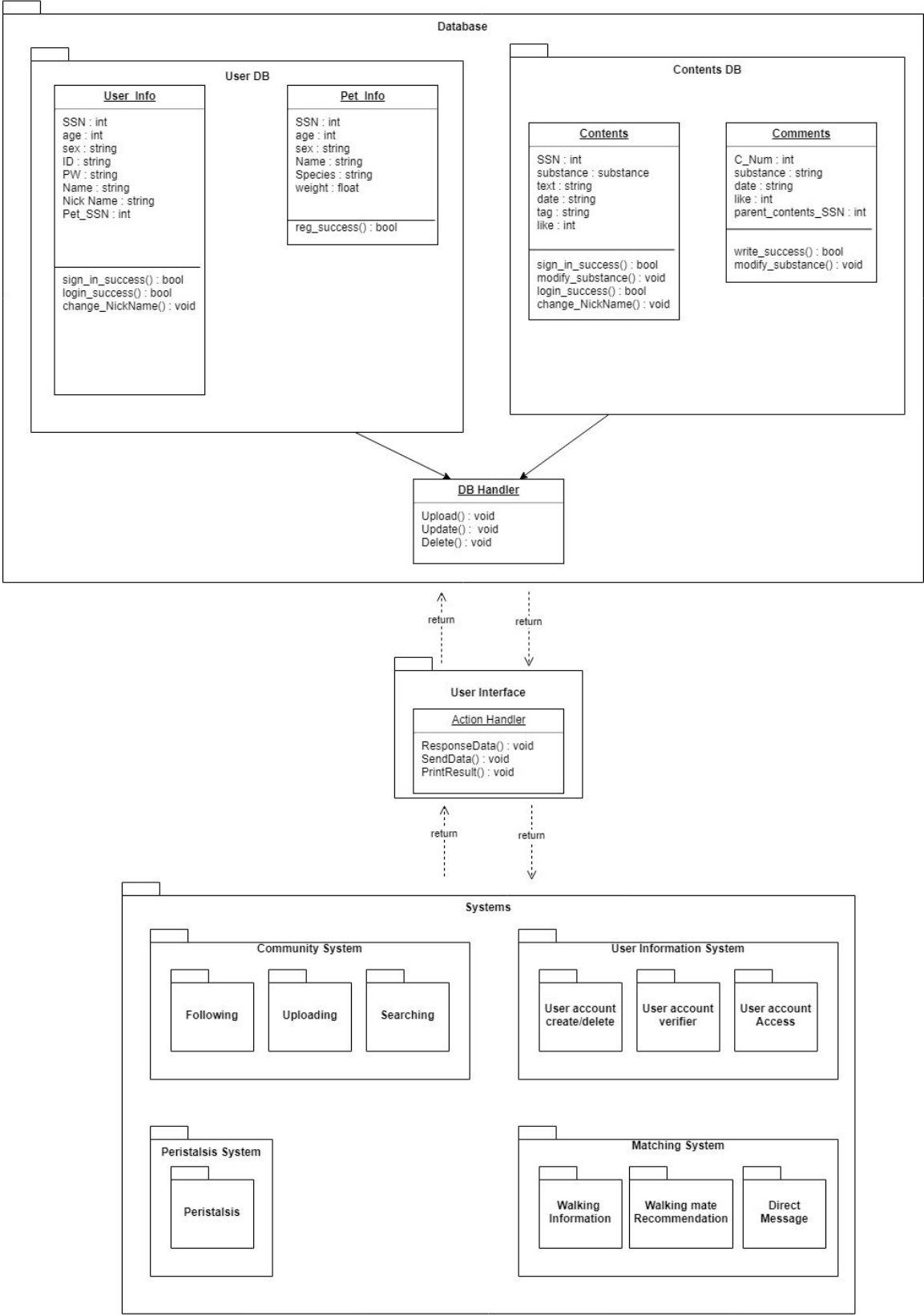
<Diagram 3.6 Context Diagram of Docking>

### 3.4 Usecase diagram



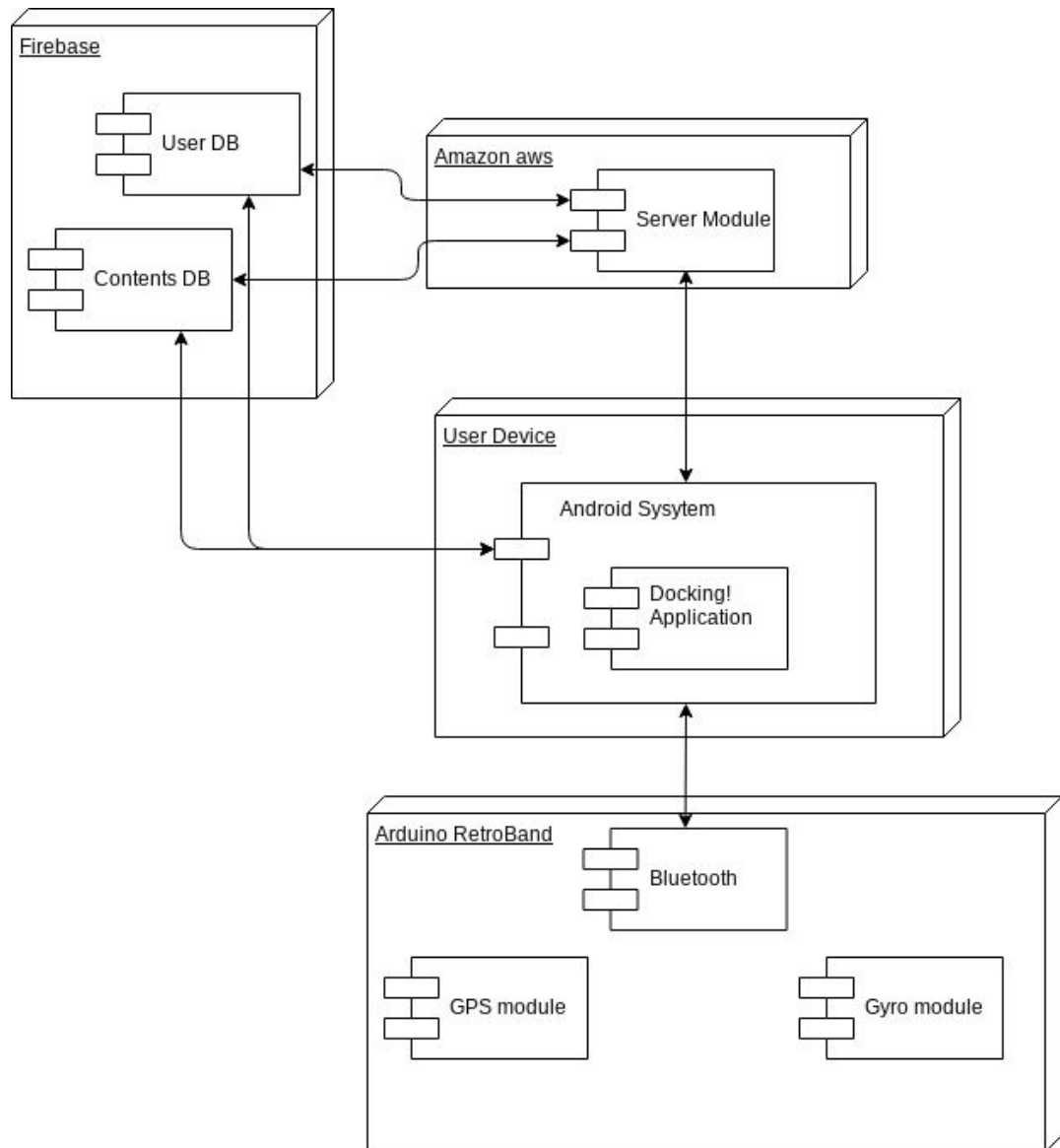
<Diagram 3.7 Use Case Diagram of Docking>

### 3.5 package diagram



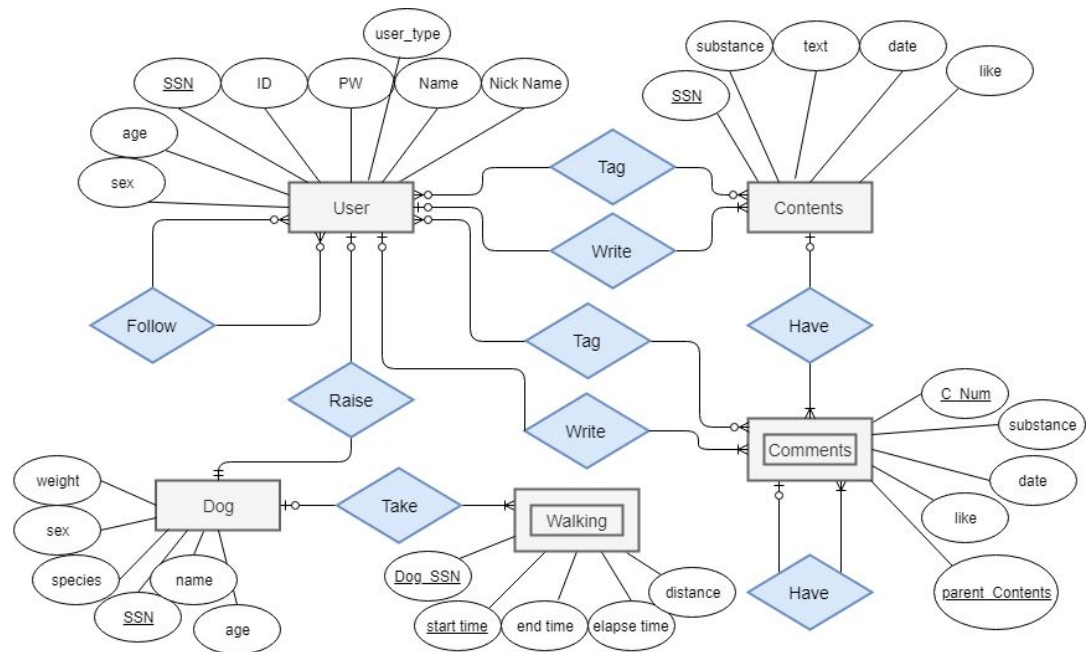
<Diagram 3.8 package Diagram of Docking>

### 3.6 deployment diagram

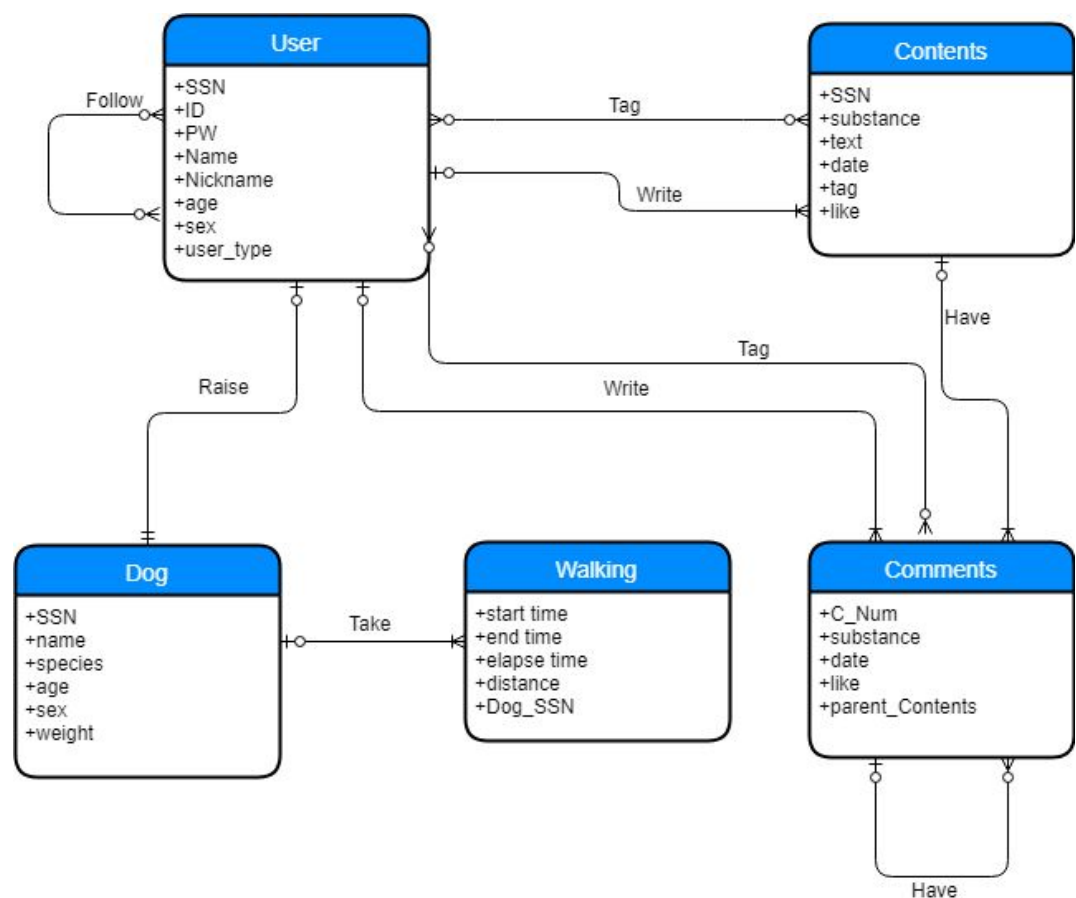


<Diagram 3.9 Deployment Diagram of Docking>

### 3.5 ER diagram



<Diagram 3.10 Docking ER Diagram>



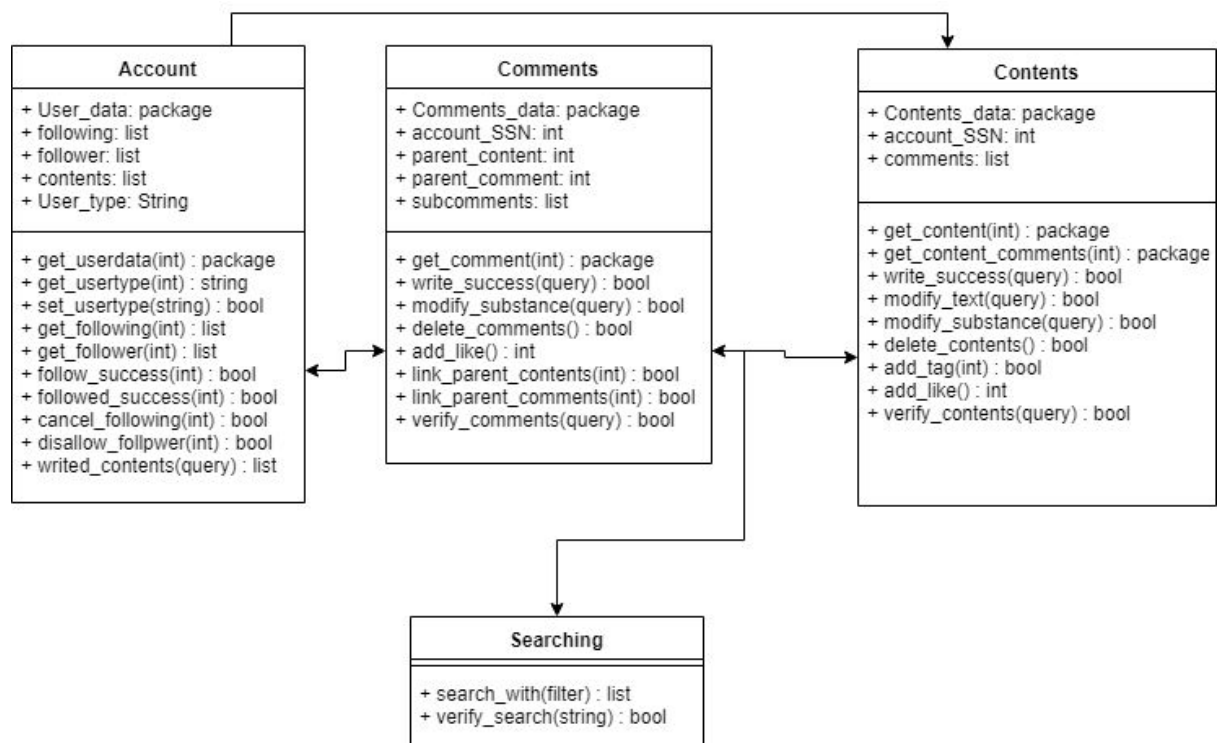
<Diagram 3.11 Docking ER Diagram ver.2>

## 4. Community System

### 4.1 Objectives

Docking에서 각 유저들이 각자 게시글을 올리고, 커뮤니티 활동을 하도록 하기 위하여 DOCKING은 사진 혹은 비디오 uploading 시스템을 제공한다. 또한 게시글을 업로드할 때 다른 계정을 태그할 수 있는 기능과 다른 유저들이 게시한 게시글 중 원하는 정보를 찾고 그 게시글에 댓글을 달 수 있도록 하는 기능 역시 제공한다. 또, 다른 유저들을 follow할 수 있는 기능을 제공하여, 유저 간 커뮤니티를 구축할 수 있도록 한다. Community System은 각 유저가 게시하는 게시글 및 유저 간 관계를 다루는 시스템으로, 이 장에서는 Community Main page, Uploading, Follow, Tag, Comments, Searching 기능을 도식화하여 서술한다.

### 4.2 Class Diagram



<Diagram 4.1 Class Diagram of Community System>

#### A. Account

##### (1)Attributes

- + User\_data: package; 사용자의 정보 패키지
- + following: list; 사용자가 팔로우하고 있는 유저 리스트
- + follower: list; 사용자를 팔로우하고 있는 유저 리스트
- + contents: list; 사용자가 작성한 contents의 고유식별번호 리스트

## (2) Methods

- + get\_userdata(int) : package; DB에서 데이터를 가져온다.
- + get\_usertype(int) : string; 사용자의 유저타입 (반려인, 비반려인, 전문가) 을 가져온다.
- + set\_usertype(string) : bool; 사용자의 유저타입 (반려인, 비반려인, 전문가) 을 설정하고 해당 요소 설정의 성패 여부를 출력한다.
- + get\_following(int) : list; 사용자가 팔로우하고 있는 유저 리스트를 가져온다.
- + get\_follower(int) : list; 사용자를 팔로우하고 있는 유저 리스트를 가져온다.
- + follow\_success(int) : bool; 사용자가 팔로우하고자 하는 유저의 고유식별번호를 input 값으로 넣어 follow의 성패 여부를 출력한다.
- + followed\_success(int) : bool; 사용자를 팔로우하길 원하는 유저의 고유식별번호를 input 값으로 넣어 follow 허가 여부를 출력한다.
- + cancel\_following(int) : bool; 사용자가 팔로우를 취소하기를 원하는 유저의 고유식별번호를 input 값으로 넣어 follow 정보를 삭제한다.
- + writed\_contents(query) : list; 사용자가 작성한 게시글 정보를 불러온다.

## B. Contents

### (1) Attributes

- + Contents\_data: package; 게시글의 정보 패키지
- + account\_SSN: int; 사용자의 고유식별번호
- + comments: list; 해당 게시글에 달린 댓글 리스트

### (2) Methods

- + get\_content(int) : package; 게시글의 고유 식별번호를 input으로 넣어 해당 게시글의 정보를 불러온다.
- + get\_content\_comments(int) : package; 게시글의 고유 식별번호를 input으로 넣어 해당 게시글이 가지고 있는 comments의 정보 패키지를 불러온다.
- + write\_success(query) : bool; 사용자가 contents를 게시하고자 하는 양식을 query로 input하여 게시의 성패 여부를 판단한다.
- + modify\_text(query) : bool; 사용자가 contents의 text를 수정하여 성패 여부를 출력한다.
- + modify\_substance(query) : bool; 사용자가 contents의 substance를 수정하여 성패 여부를 출력한다.
- + delete\_contents() : bool; 사용자가 contents를 삭제하여 성패 여부를 출력한다.
- + add\_tag(int) : bool; 사용자가 contents에 tag를 추가하여 성패 여부를 출력한다.
- + add\_like() : int; 사용자가 contents에 like를 추가하여 ++1한 값을 출력한다.
- + verify\_contents(query) : bool; 사용자가 게시하고자 하는 contents의 세부정보가 제약사항을 지키고 있는지 verify하여 그 여부를 출력한다.

## C. Comments

### (1) Attributes

- + Comments\_data: package; 사용자의 댓글 데이터 패키지
- + account\_SSN: int; 사용자의 고유식별번호
- + parent\_content: int; 해당 댓글이 속해 있는 content의 고유식별번호
- + parent\_comment: int; 해당 댓글이 속해 있는 comment의 고유식별번호
- + subcomments: list; 해당 댓글에 달린 대댓글 리스트



(2) Methods

- + get\_comment(int) : package; 댓글의 고유식별번호를 input으로 넣어 comment의 정보 패키지를 불러온다.
- + write\_success(query) : bool; 사용자가 댓글을 입력하여 성패 여부를 출력한다.
- + modify\_substance(query) : bool; 사용자가 댓글의 내용을 수정하여 성패 여부를 출력한다.
- + delete\_comments() : bool; 사용자가 댓글을 삭제하여 성패 여부를 출력한다.
- + add\_like() : int; 사용자가 댓글에 like를 추가하여 ++1한 값을 출력한다.
- + link\_parent\_contents(int) : bool; 해당 댓글과 해당 댓글이 속해 있는 content를 연결한 뒤 DB에 저장하여 성패 여부를 출력한다.
- + link\_parent\_comments(int) : bool; 해당 댓글과 해당 댓글이 속해 있는 comment를 연결한 뒤 DB에 저장하여 성패 여부를 출력한다.
- + verify\_comments(query) : bool; 사용자가 입력하고자 하는 댓글이 제약사항을 만족하는지 여부를 출력한다.

## D. Searching

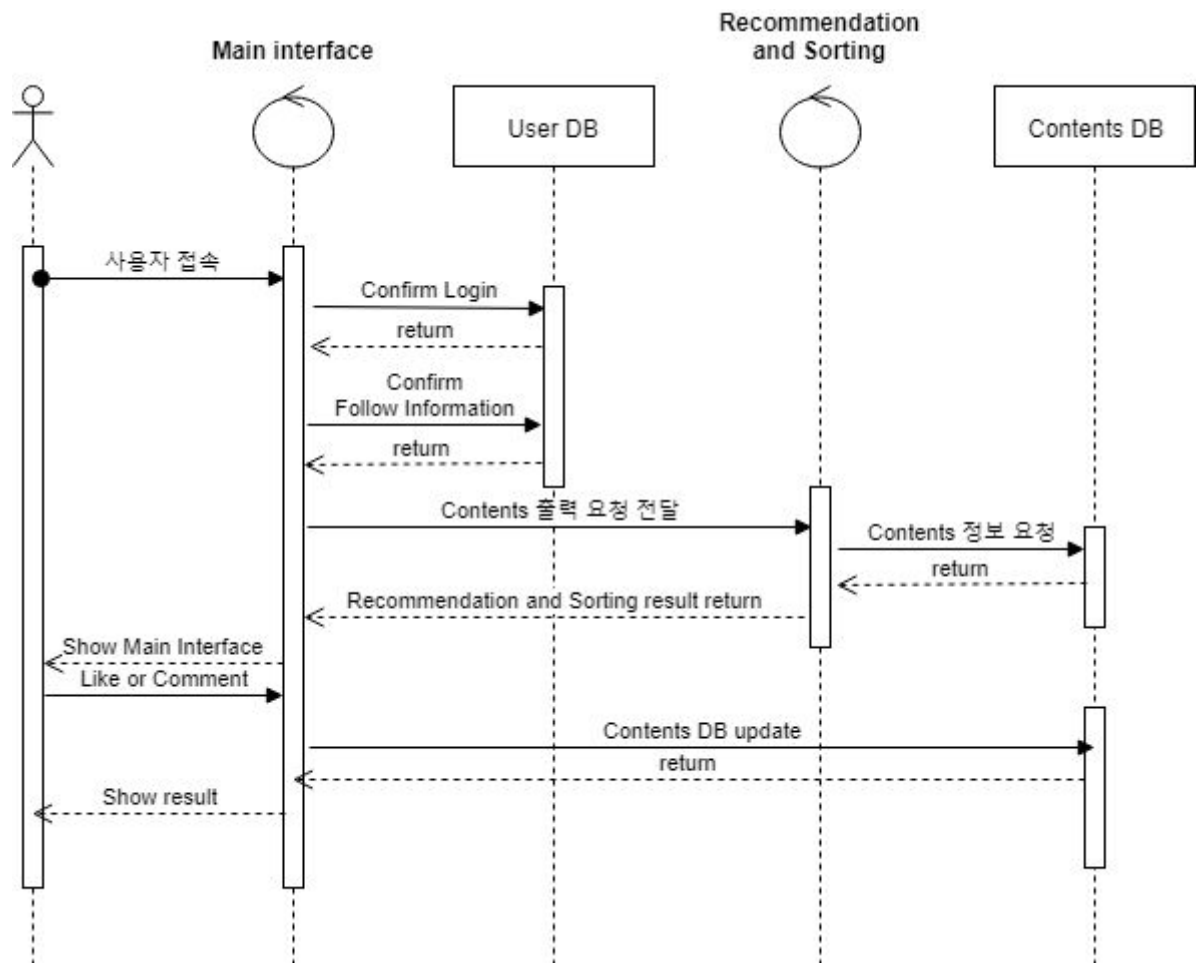
(1) Attributes : 해당없음

(2) Methods

- + search\_with(filter) : list; filter값에 text, user, 또는 tag 데이터를 input으로 넣고 서치 결과를 출력한다.
- + verify\_search(string) : bool; search 양식이 제약사항을 준수하고 있는지 확인한다.

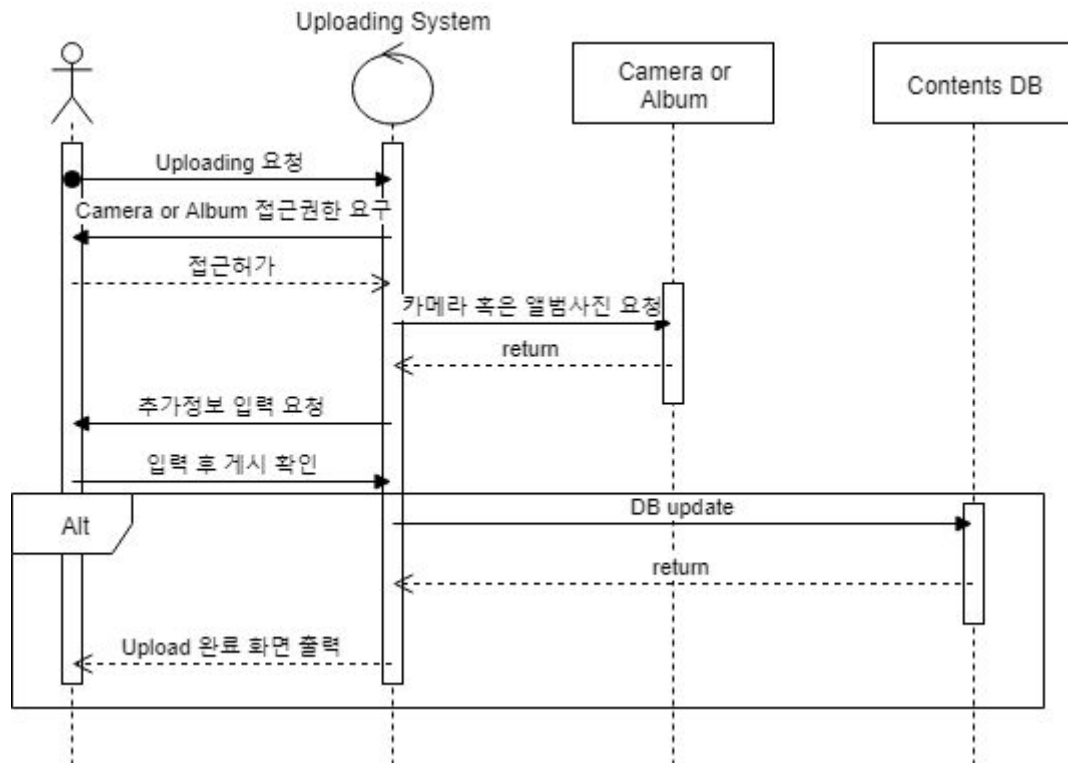
### 4.3 Sequence Diagram

#### A. Community Main page



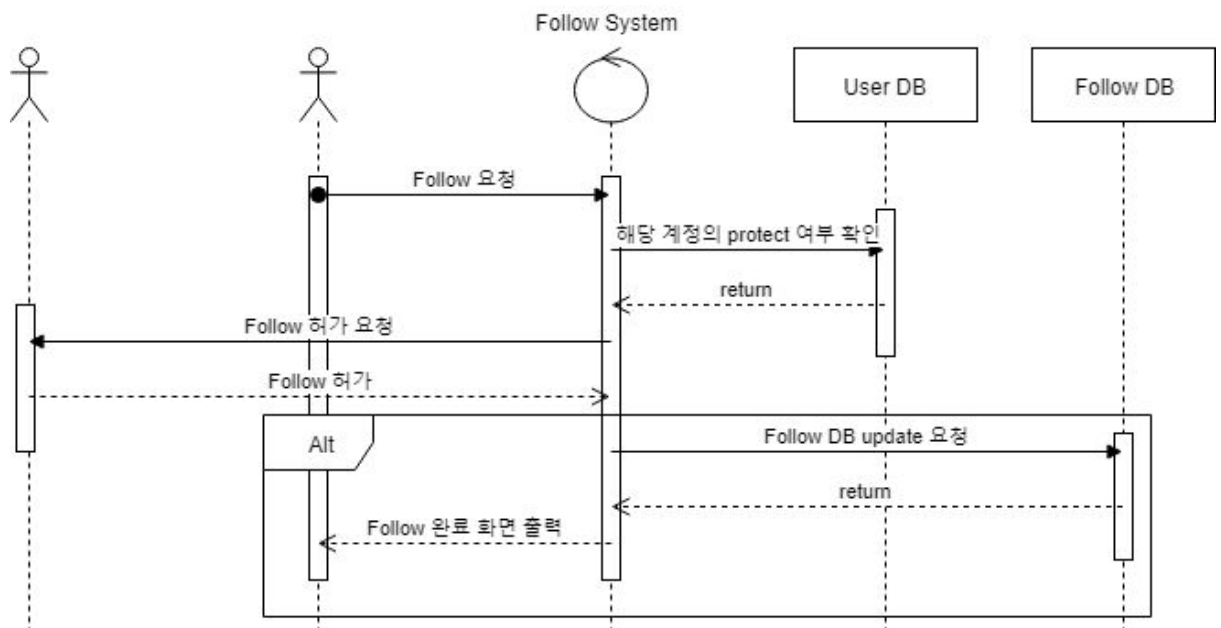
<Diagram 4.2 Community Main page Sequence Diagram>

## B. Uploading



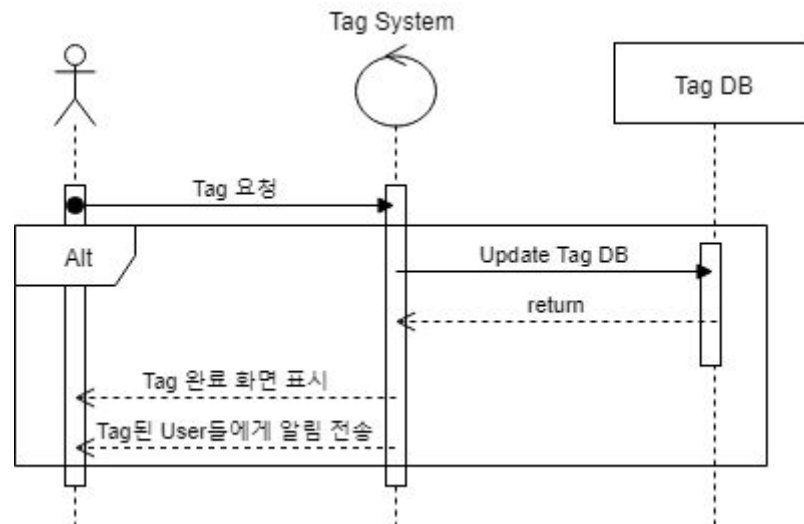
<Diagram 4.3 Uploading Sequence Diagram>

## C. Follow



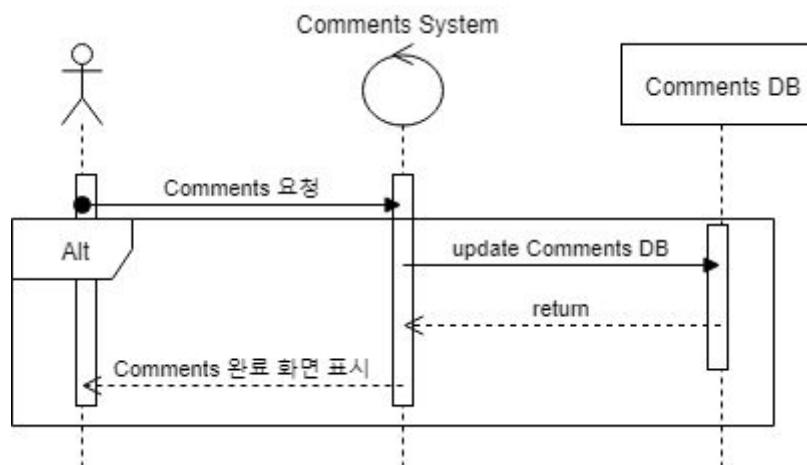
<Diagram 4.4 Follow Sequence Diagram>

## D. Tag



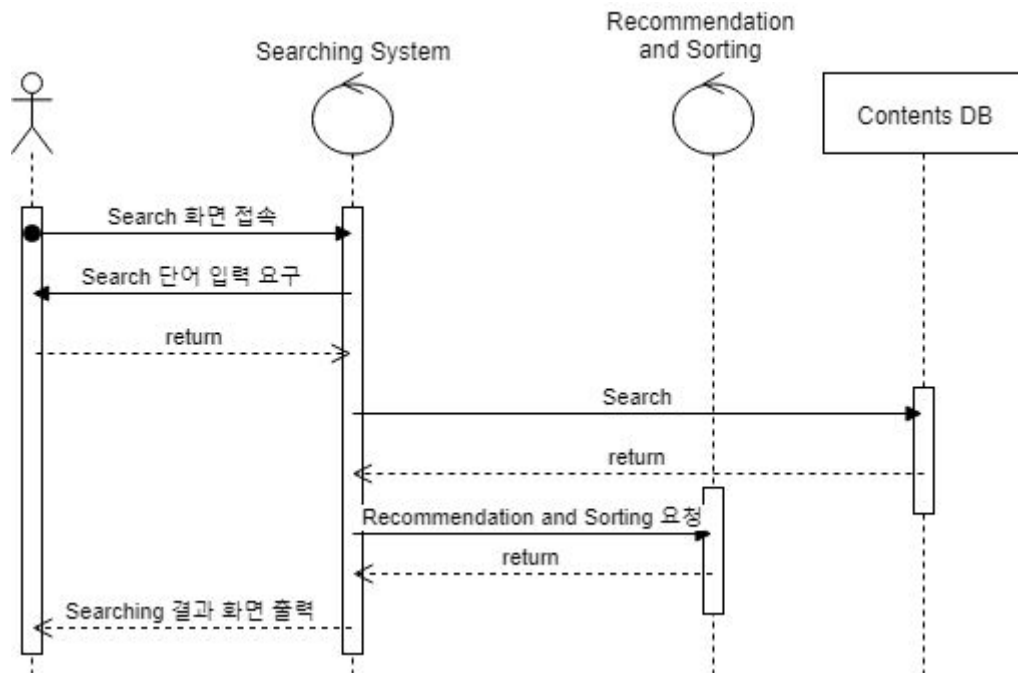
<Diagram 4.5 Tag Sequence Diagram>

## E. Comments



<Diagram 4.6 Comments Sequence Diagram>

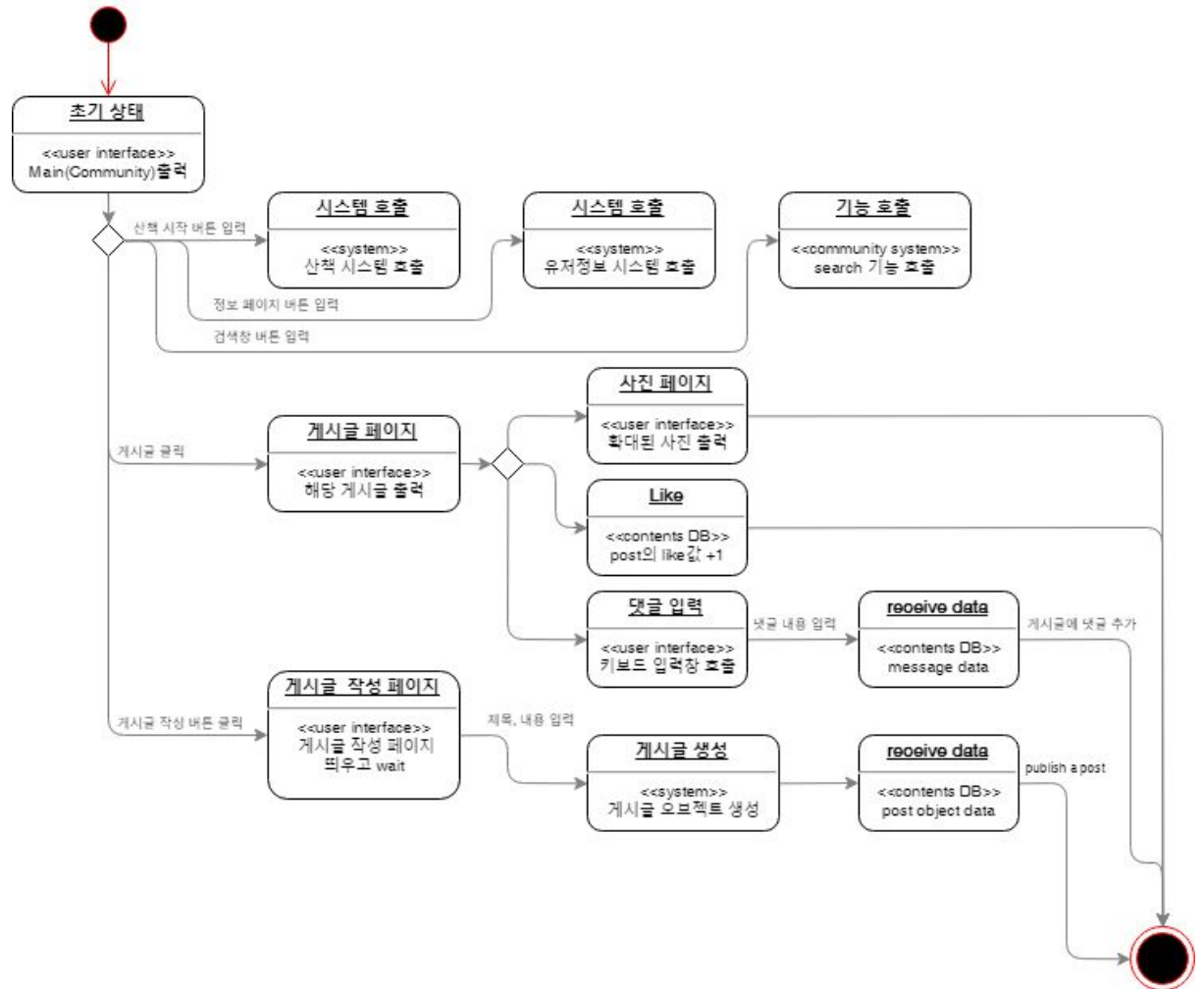
## F. Searching



<Diagram 4.7 Searching Sequence Diagram>

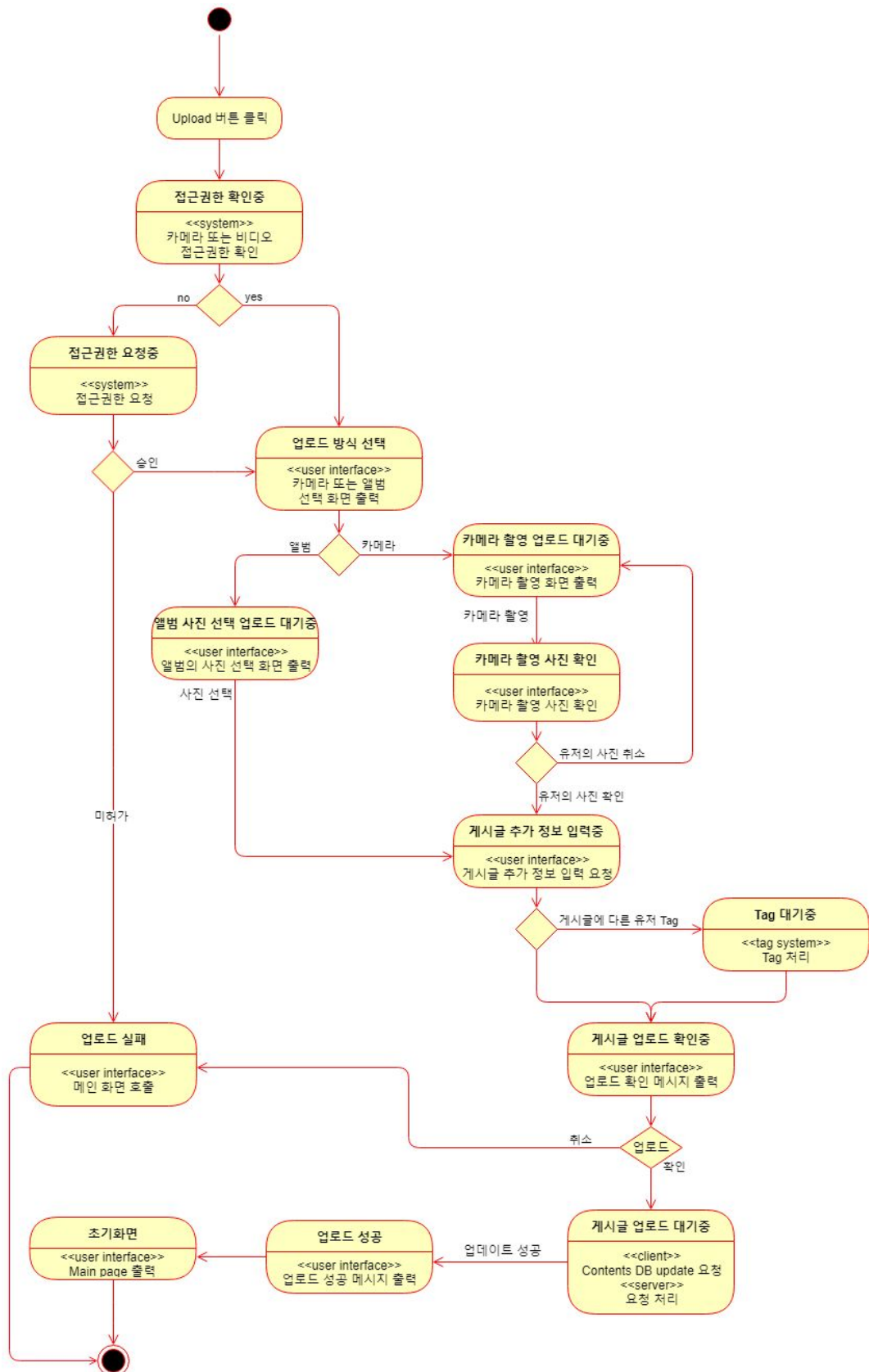
## 4.4 State Diagram

### A. Community Main page



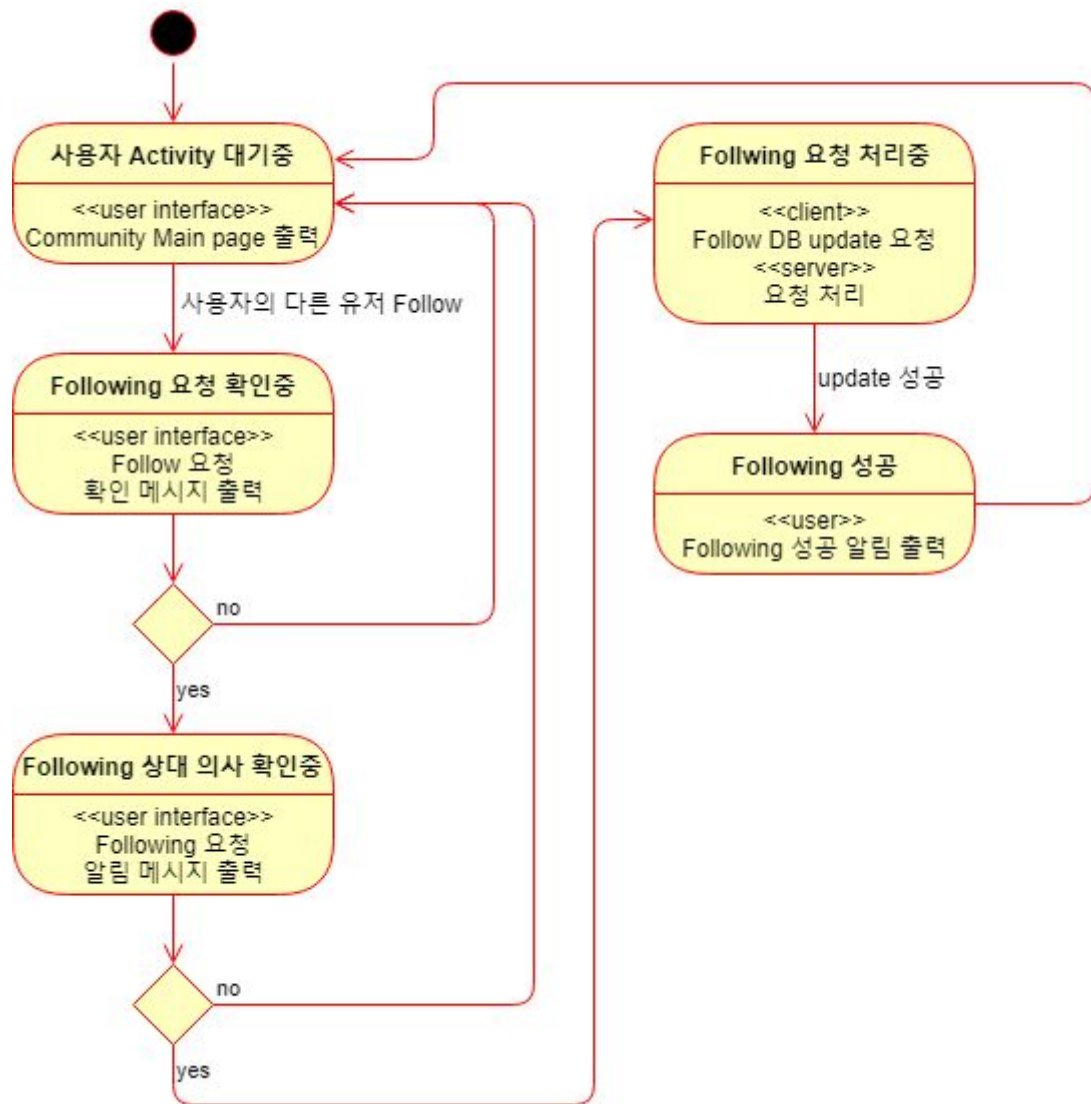
〈Diagram 4.8 Community main page State Diagram〉

## B. Uploading



〈Diagram 4.9 Uploading State Diagram〉

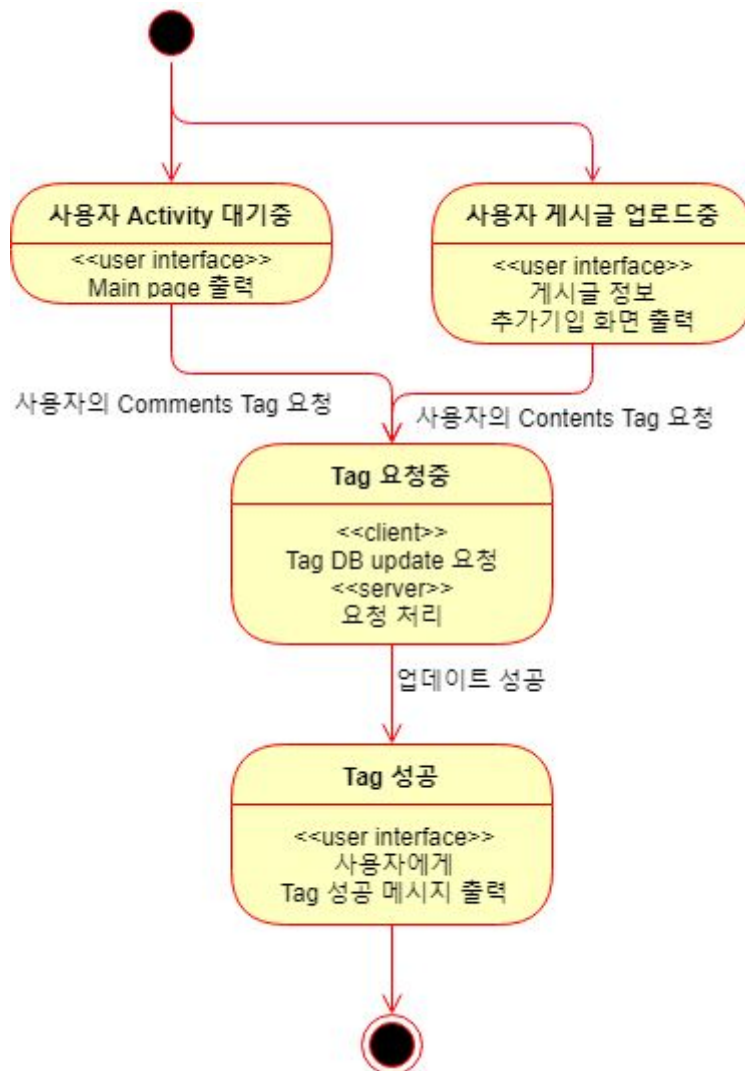
## C. Follow



〈Diagram 4.10 Follow State Diagram〉

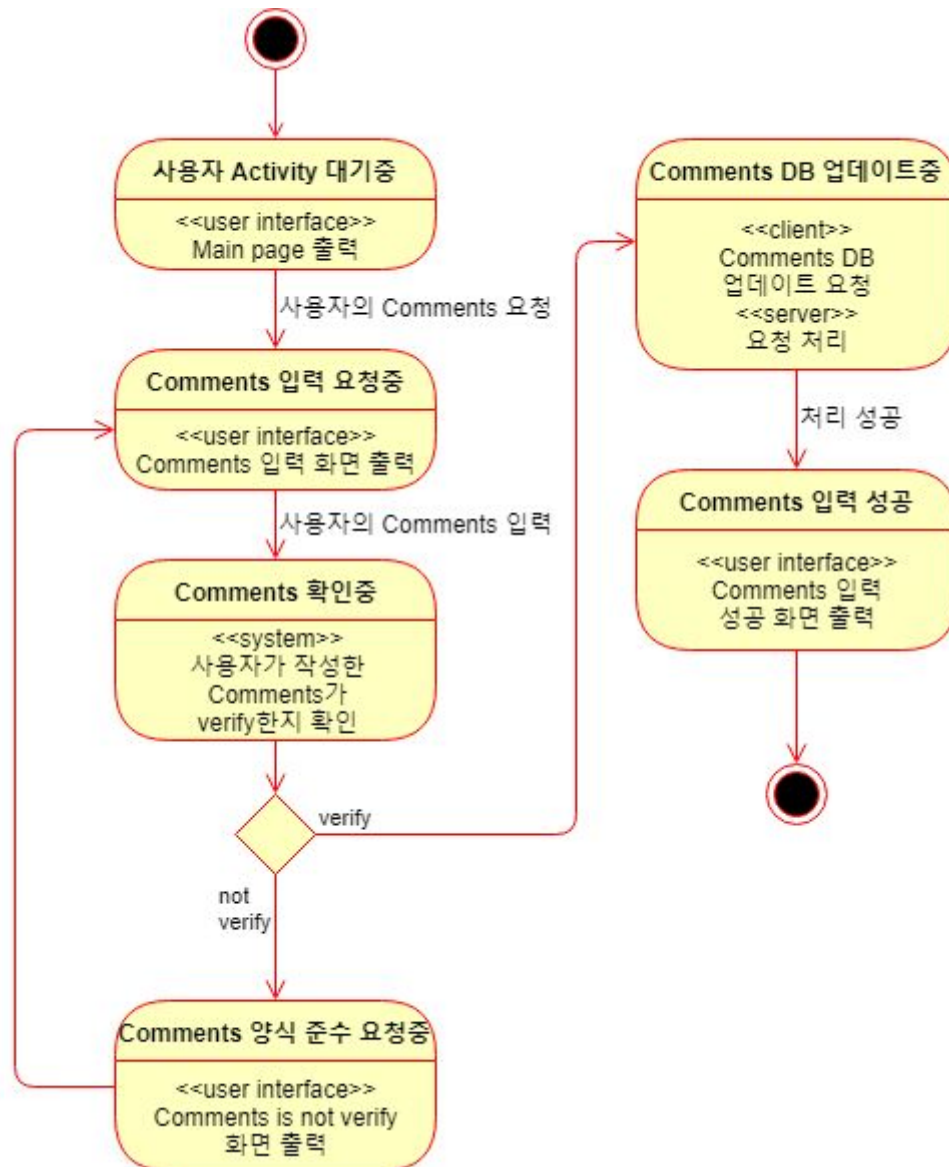


## D. Tag



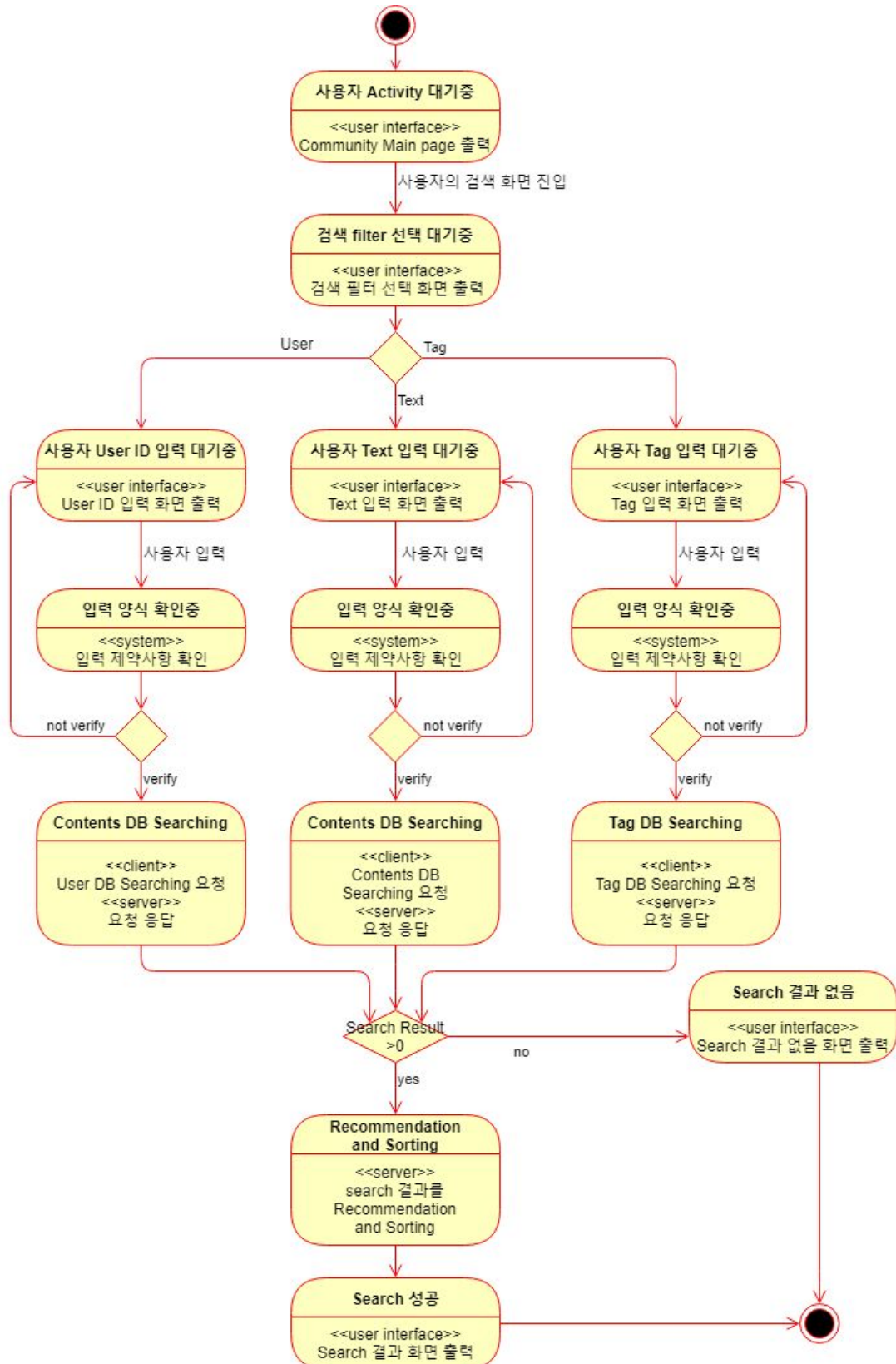
〈Diagram 4.11 Tag State Diagram〉

## E. Comments



<Diagram 4.12 Comments State Diagram>

## F. Searching



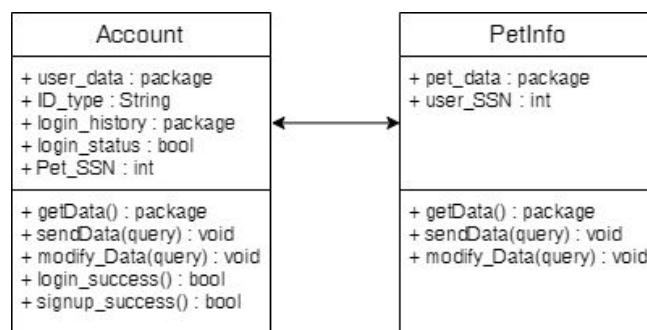
〈Diagram 4.13 Searching State Diagram〉

## 5 User Information System

### 5.1 Objectives

DOCKING의 커뮤니티 시스템 사용을 위해 각 유저는 개인 계정을 필요로 하며, DOCKING은 개인 계정을 통해 유저에게 팔로우, 마이페이지, 산책기록 조회 등의 추가 기능을 제공한다. User Information System은 각 계정에 포함되는 정보들을 다루는 시스템으로, 이 장에서는 회원가입, 로그인, 계정 정보에 접근하는 기능을 도식화하여 서술한다.

### 5.2 Class Diagram



<Diagram 5.1 Class Diagram of User Information System>

#### A. Account

##### (1) Attributes

- + user\_data : package : 사용자의 정보 패키지
- + ID\_type : String : 사용자의 아이디가 네이버 아이디일 경우 api 키값
- + login\_history : package : 사용자의 로그인 기록
- + login\_status : bool : 로그인 상태
- + Pet\_SSN : int : 반려견의 고유식별번호

##### (2) Methods

- + getData() : package : DB에서 data를 가져온다.
- + sendData(query) : void : DB로 data를 보낸다.
- + modify\_Data(query) : void : 수정된 data를 DB로 보낸다
- + login\_success() : bool : 로그인이 성공했을 경우 true 리턴
- + signup\_success() : bool : 계정 생성에 성공했을 경우 signup 리턴

#### B. PetInfo

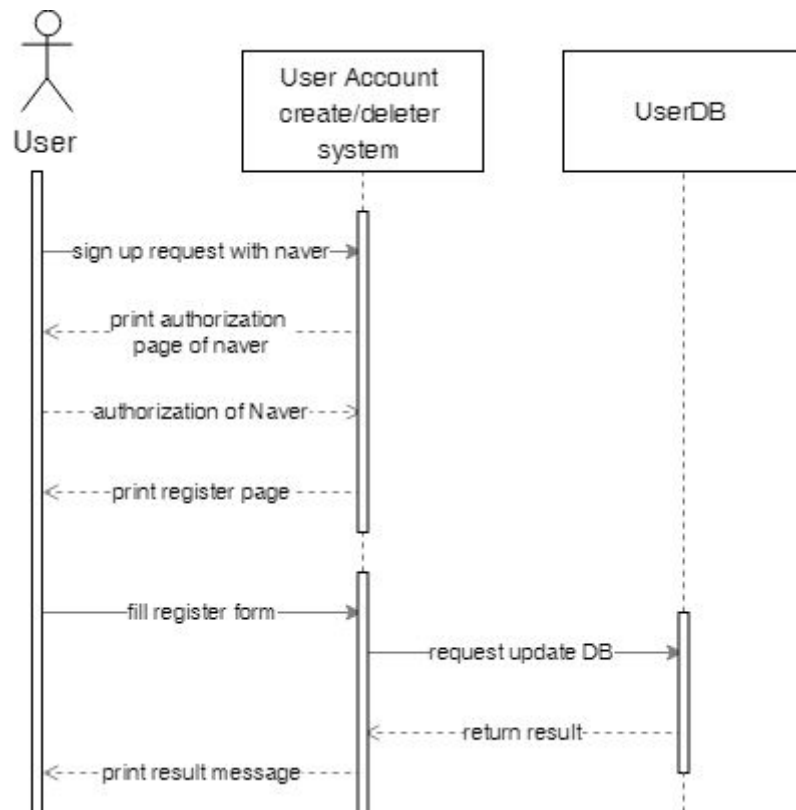
##### (1) Attributes

- + pet\_data : package : 반려견의 정보 패키지

- + user\_SSN : int : 해당 반려건의 견주인 유저의 고유식별번호
- (2) Methods
- + getData() : package : DB에서 data를 가져온다.
- + sendData(query) : void : DB로 data를 보낸다.
- + modify\_Data(query) : void : 수정된 data를 DB로 보낸다

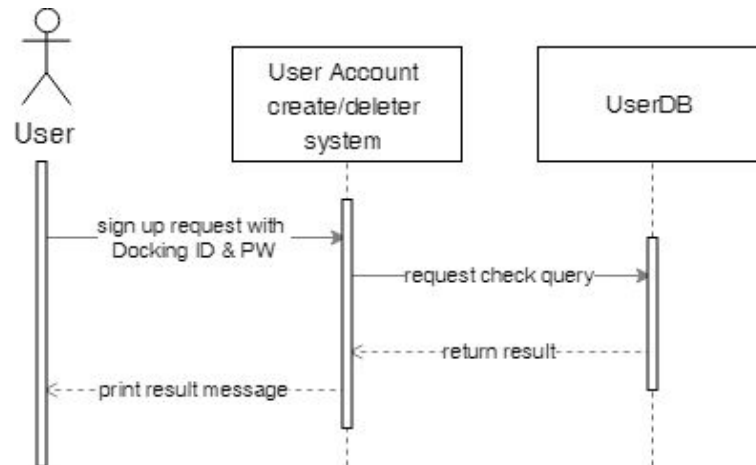
## 5.3 Sequence Diagram

### A. Sign up with Naver



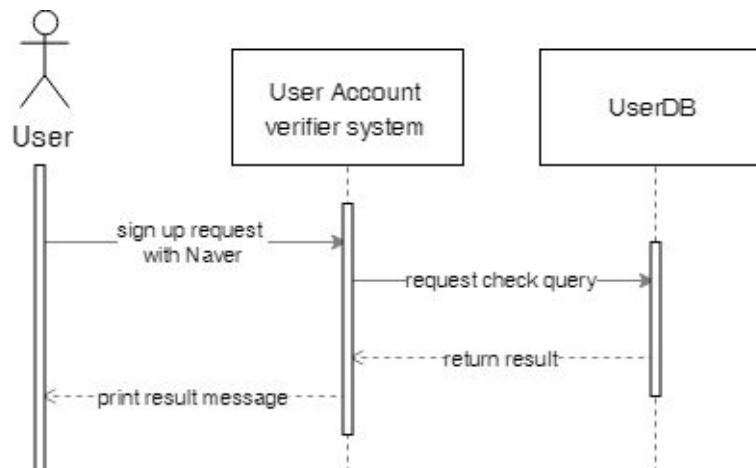
<Diagram 5.2 Sequence diagram of Sign up with Naver >

## B. Sign up with Docking



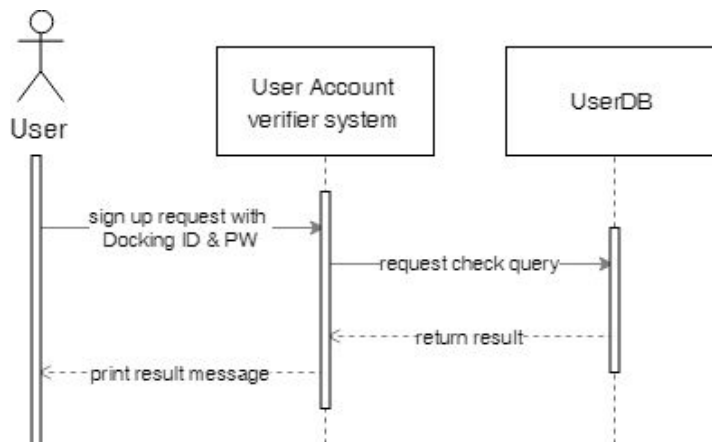
<Diagram 5.3 Sequence diagram of Sign up with Docking >

## C. Log in with Naver



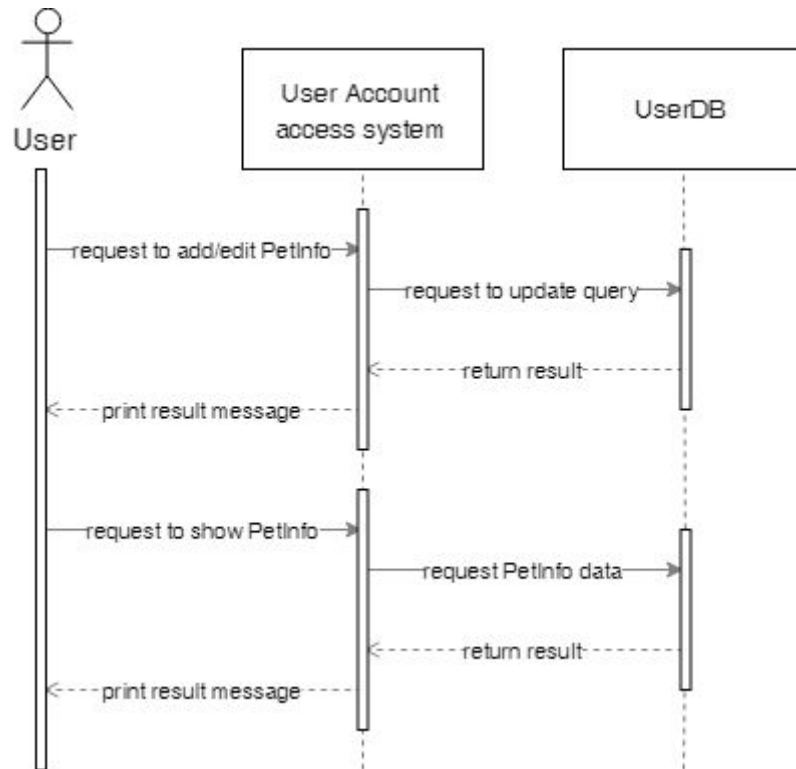
<Diagram 5.4 Sequence diagram of Log in with Naver >

## D. Log in with Docking



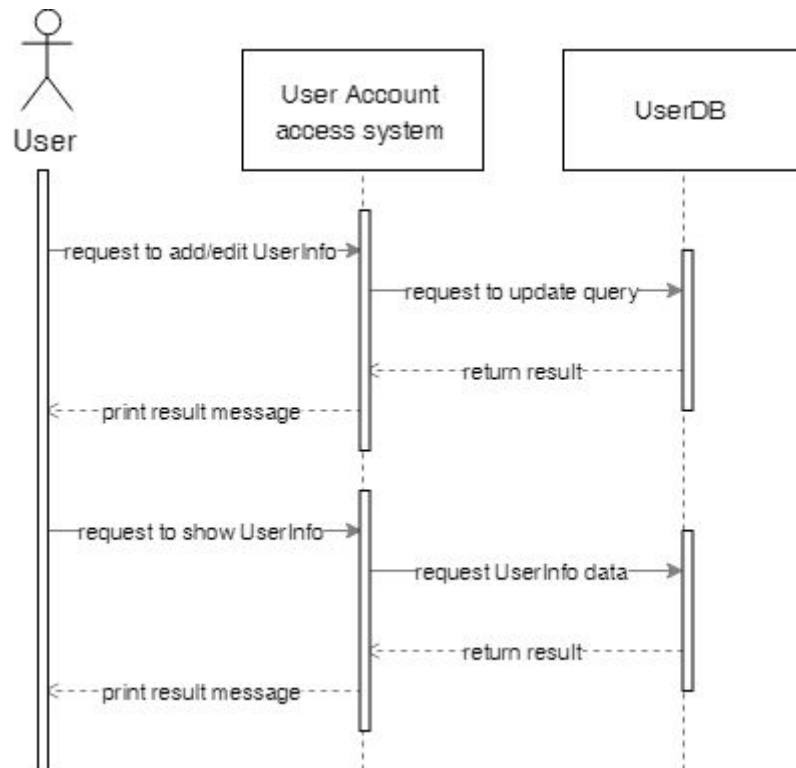
<Diagram 5.5 Sequence diagram of Log in with Docking >

## E. Dog List Management



<Diagram 5.6 Sequence diagram of Dog List Management >

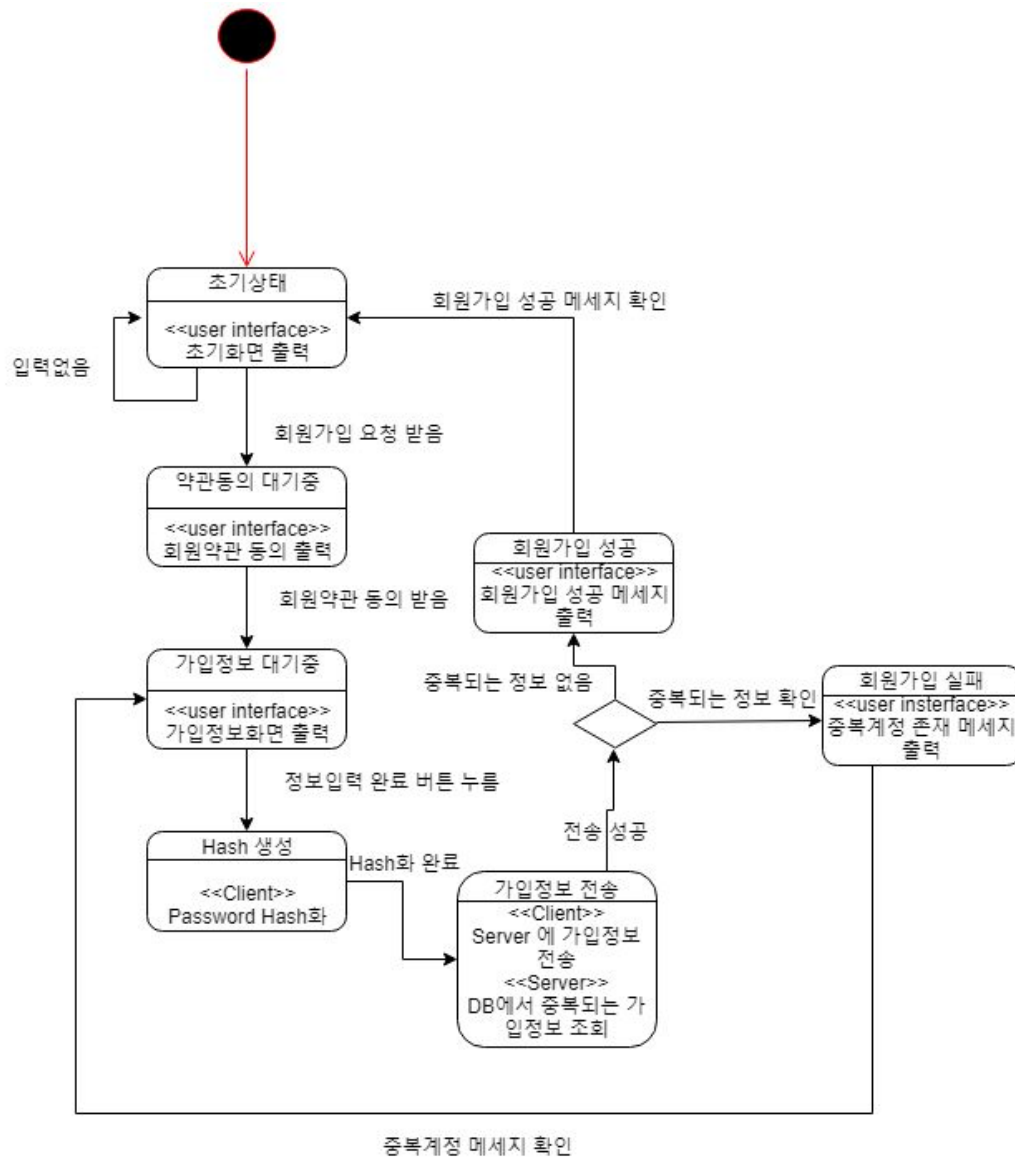
## F. User Account Management



<Diagram 5.7 Sequence diagram of User Account Management >

## 5.4 State Diagram

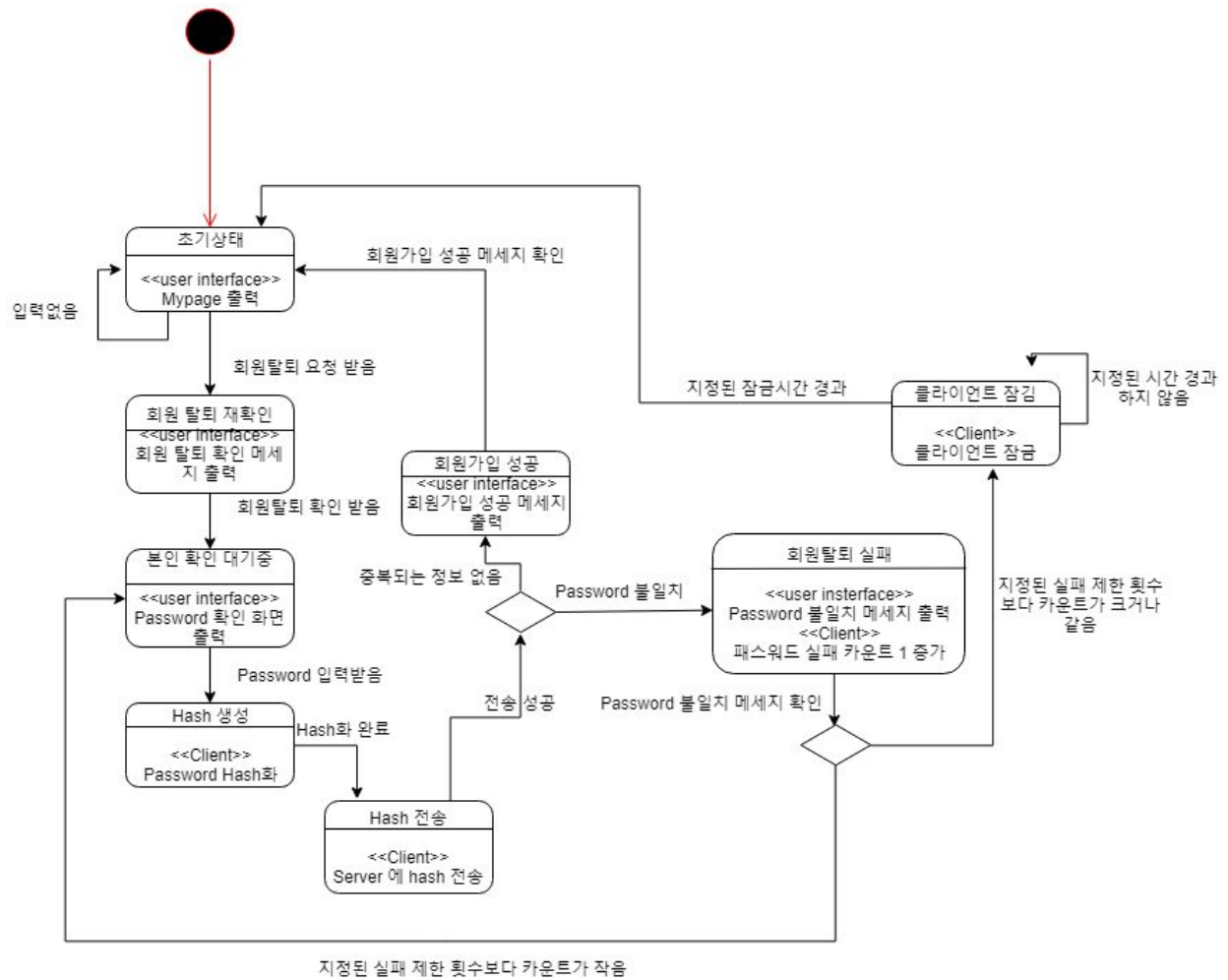
### A. Sign in



<Diagram 5.8 State diagram of Sign in>

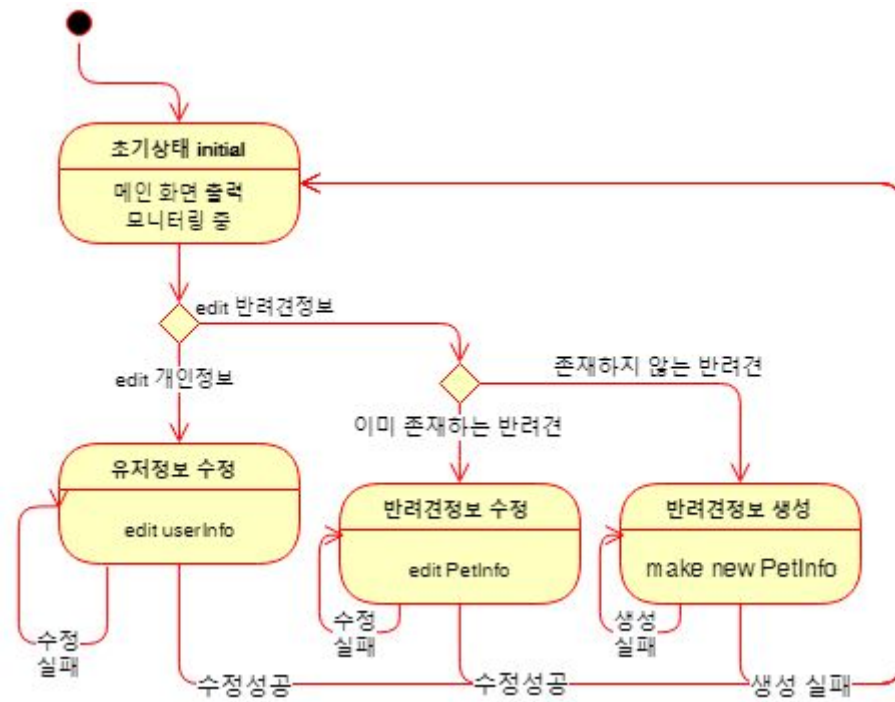


## B. Sign out



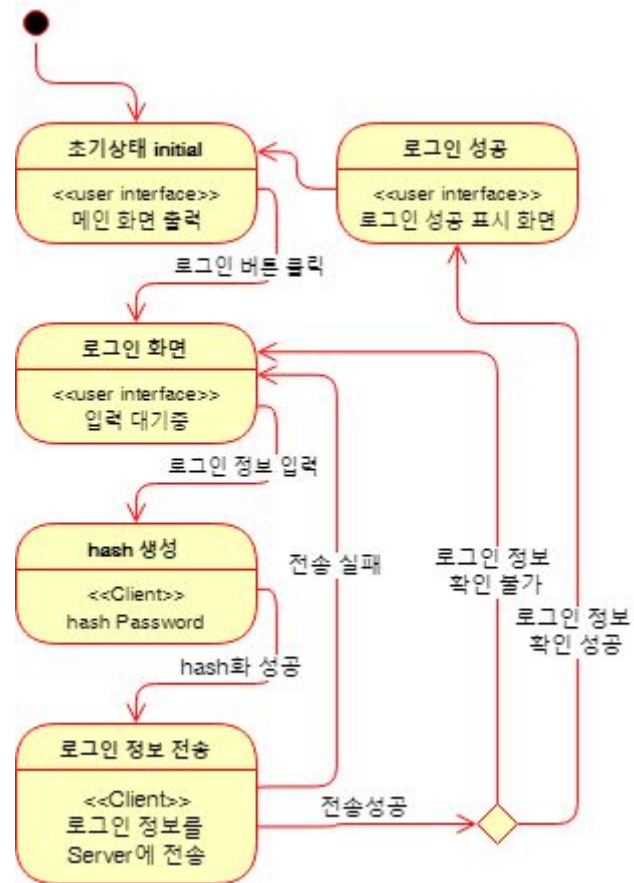
<Diagram 5.9 State diagram of Sign out>

### C. Log in



〈Diagram 5.10 State diagram of Log in〉

## D. User account Access



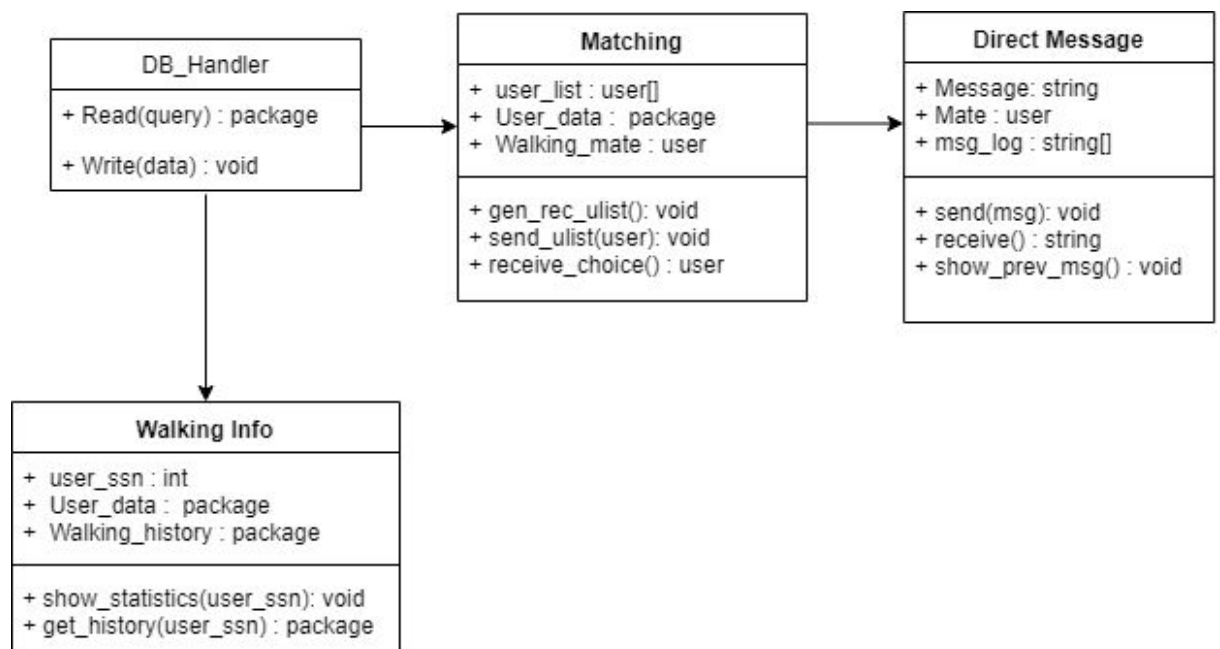
〈Diagram 5.11 State diagram of User Account Access〉

## 6 . Matching System

### 6.1 Objective

산책 Mate를 매칭시켜주는 시스템의 설계를 설명한다. Mate 매칭은 유저DB에 저장된 정보에 따라 적절한 Mate를 추천시켜주고, 그 Mate와의 연락을 돕는 Direct Message subsystem과 산책정보, 통계를 보여주는 Walking Information subsystem 으로 구성된다. 사용자가 혼자 산책할지 아니면 Mate 와 matching을 원하는지에 따라 상황이 달라지기 때문에, 그것을 case로 설정하여 Class diagram, Sequence diagram, State Diagram을 통해 Matching System의 구조와 user간의 상호 작용을 표현하고 설명한다.

### 6.2 Class Diagram



<Diagram 6.1 Class diagram of Matching System>

#### A. DB Handler

(1) Attributes

해당 사항 없음.

(2) Methods

- + package Read(query) : DB에서 원하는 data package를 읽어온다.
- + void Write(data) : 해당 data를 DB에 저장한다.
- + DB Handler

#### B. Matching

(1) Attributes

- + user\_list : 추천 user 들의 list.
- + User\_data : Matching 을 원하는 user의 정보.
- + Walking\_mate : 최종 matching된 유저.

(2) Methods

- + user receive\_choice() : 사용자로부터 user 선택정보를 받아옴.
- + void send\_ulist(user) : 해당 추천유저들의 list를 사용자에게 전송한다.
- + void gen\_rec\_ulist() : user 정보를 바탕으로 추천 list를 생성한다.

## C. Direct Message

(1) Attributes

- + Message : 상대방과 송/수신하는 메시지.
- + Mate : 상대방 mate 정보.

(2) Methods

- + void send(msg) : 상대방에게 메시지 송신.
- + string receive() : 상대방으로부터 메시지 수신.

## D. Walking Info

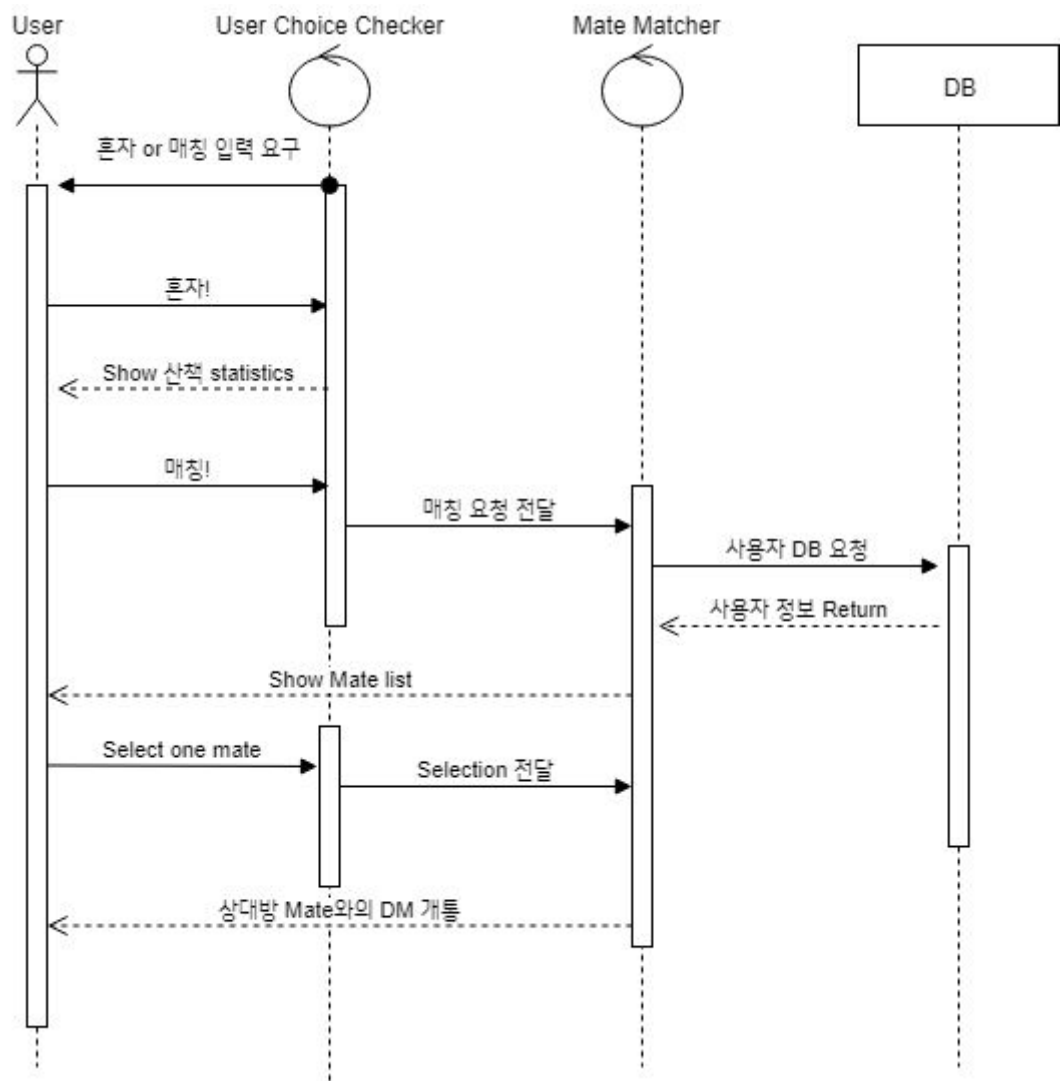
(1) Attributes

- + user\_ssn : user의 ssn.
- + User\_data : 사용자의 data package.
- + Walking\_history : 사용자의 산책 history정보.

(2) Methods

- + void show\_statistics(user\_ssn) : 해당 유저의 현재산책 통계 show.
- + package get\_history(user\_ssn) : 해당 유저의 과거산책 history 를 받아온다.

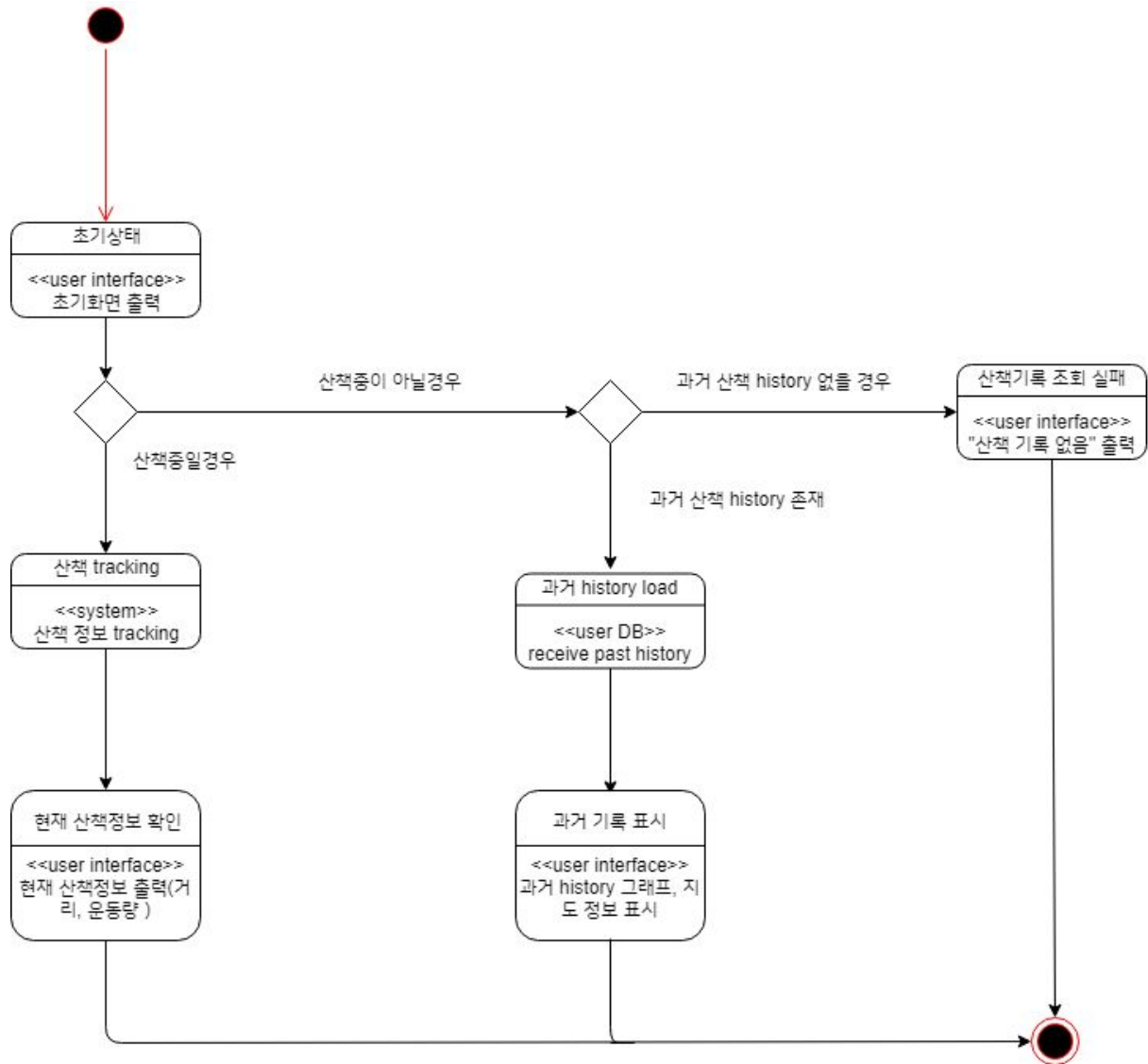
### 6.3 Sequence Diagram



<Diagram 6.2 Sequence diagram for matching system>

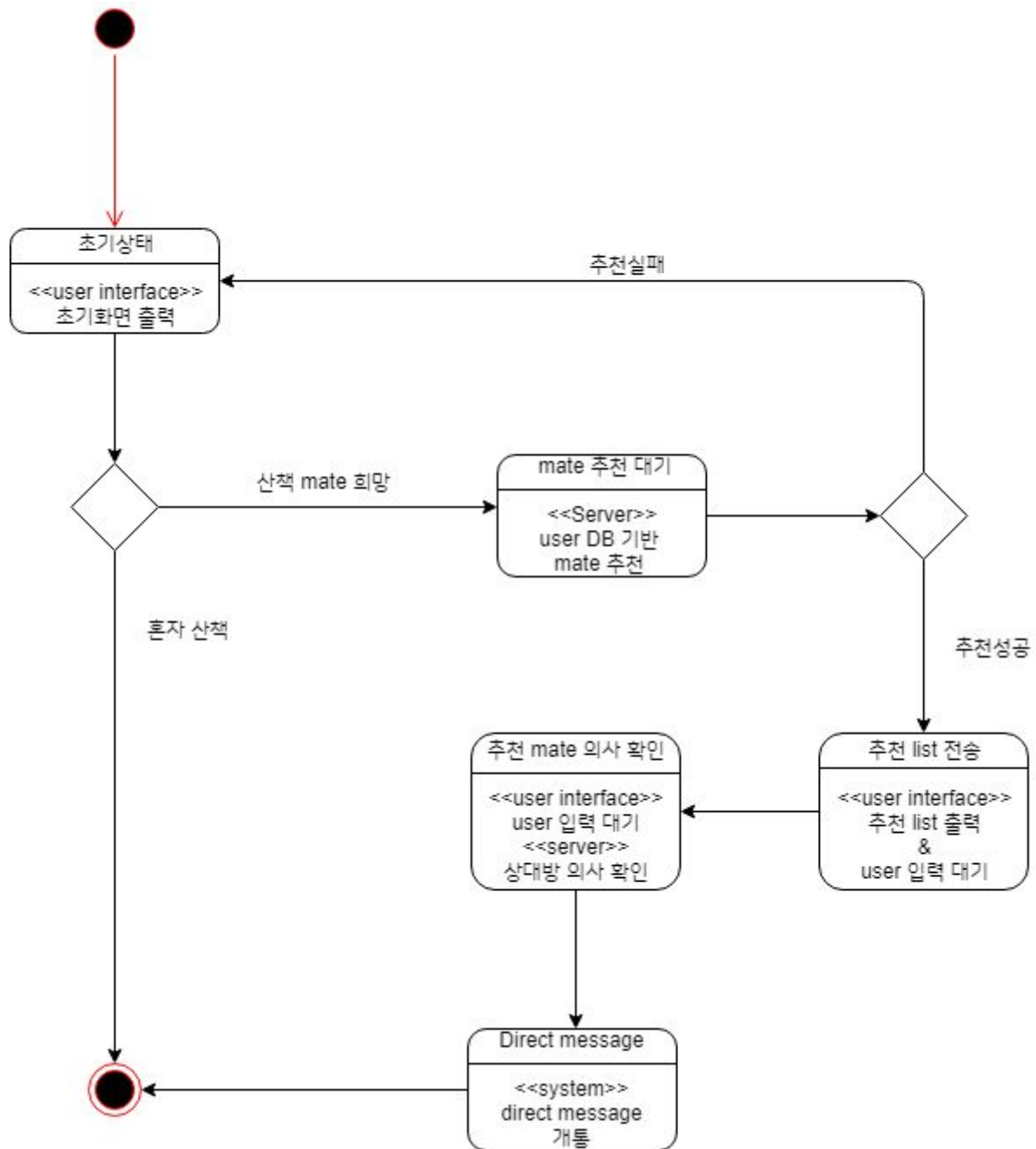
## 6.4 State Diagram

### A. Walking Info



<Diagram 6.3 State diagram of Walking Info>

## B. Walking Mate Matching & Direct Message



<Diagram 6.4 State diagram of Mate matching & DM>

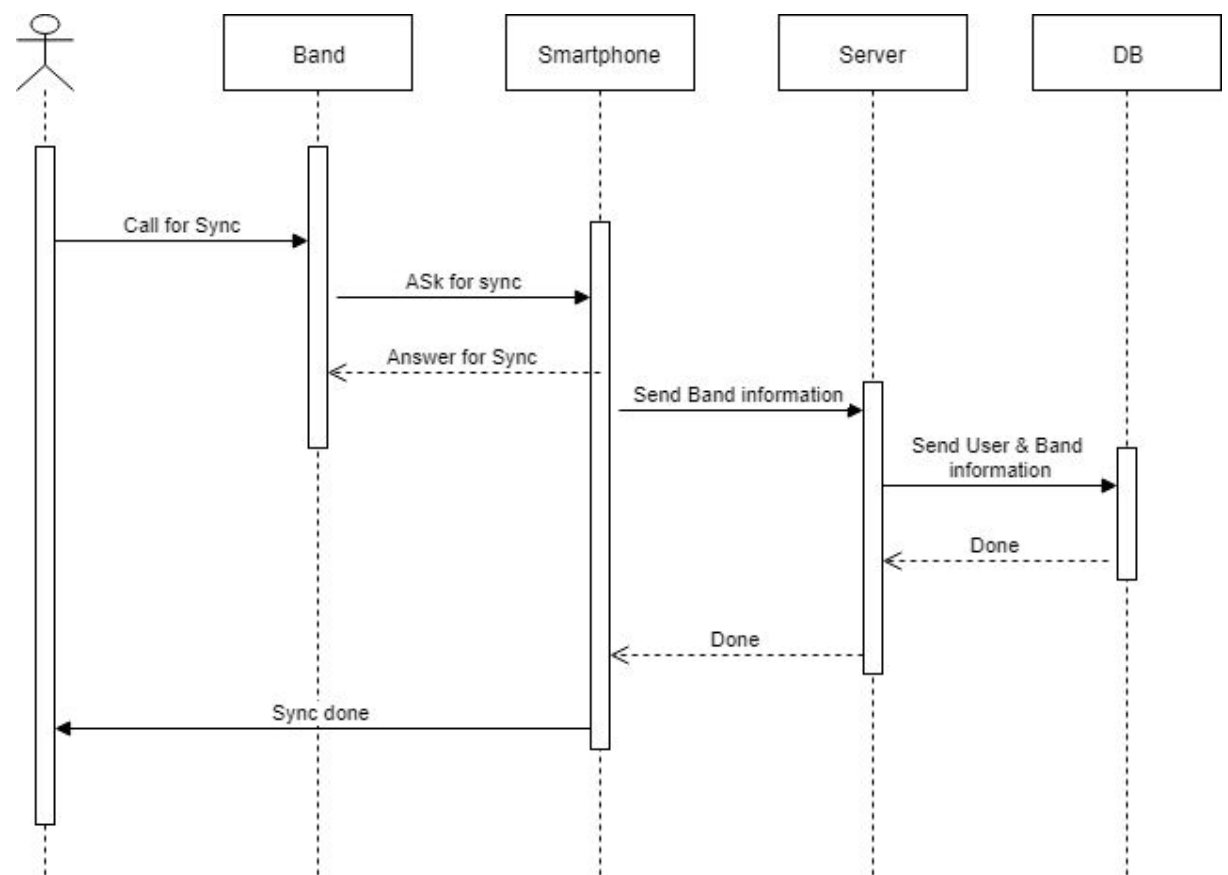


## 7 Peristalsis System

### 7.1 Objective

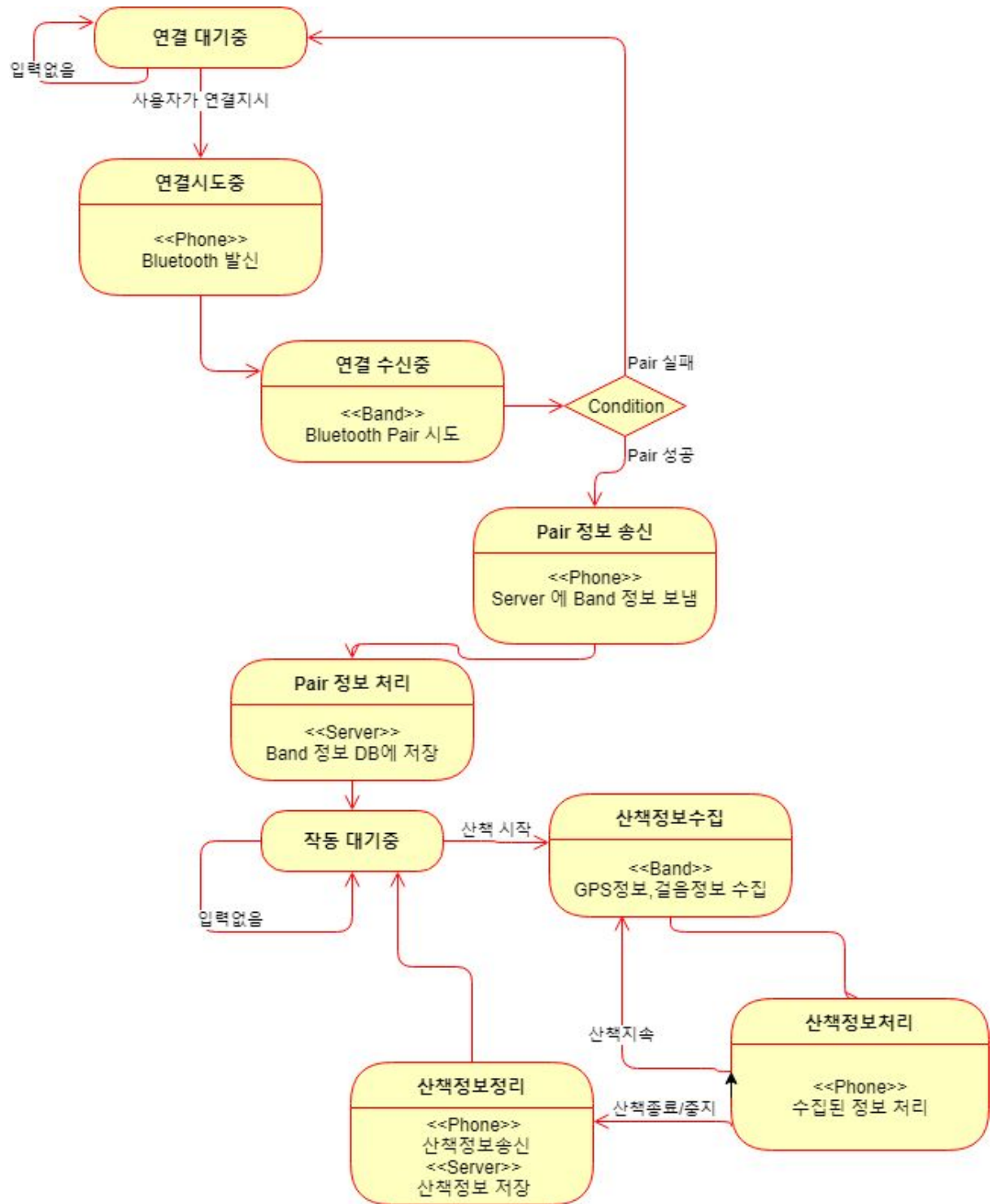
프로토타입 스마트 밴드는 오픈소스 프로젝트인 'Retroband'에 GPS 모듈을 추가하여 사용한다. 프로토타입 스마트밴드의 제어소체는 Arduino NANO를 사용하며 가속도센서는 3축 가속도센서인 MPU-6050를 사용한다. 사용자의 스마트폰과 통신하는 통신모듈은 블루투스 모듈인 HC-06 혹은 HM-10을 사용한다.

### 7.2 Band Sync Sequence Diagram



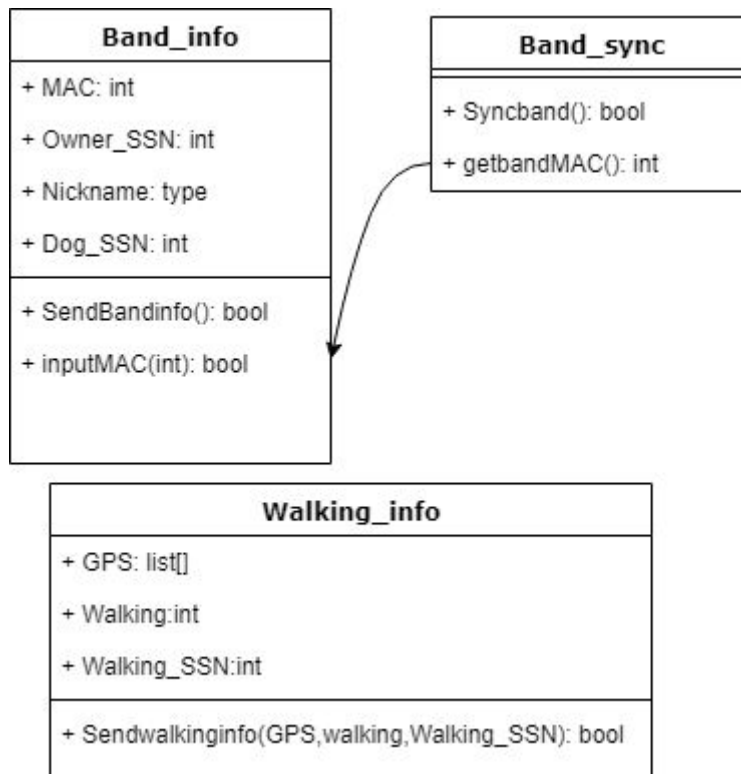
<Diagram 7.1 Sequence Diagram of Band Sync>

### 7.3 Band working State diagram



<Diagram 7.2 State Diagram of Band>

### 7.4 Class Diagram



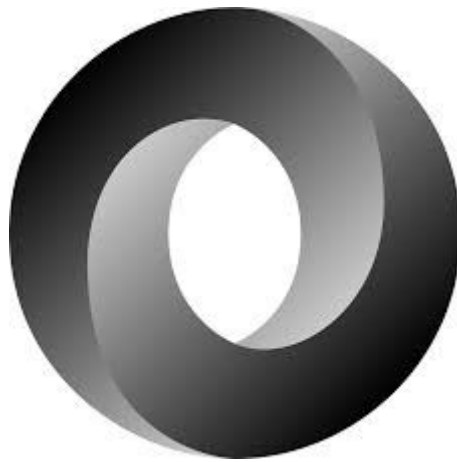
<Diagram 7.3 Class Diagram of Band>

## 8 Protocol Design

### 8.1 Objectives

이 장에서는 DOCKING에서 각 서브 시스템 간의 communication에 있어 필요한 protocol에 대해 기술하고자 한다. Protocol에 필요한 data format으로는 XML, JSON등이 있으며, 본 시스템에서는 JSON을 사용한다. 따라서 이 장에 관련한 설명과 각 protocol에서 사용되는 Attribute와 Value는 무엇인지 자세하게 서술한다.

### 8.2 JSON



<Figure 8.1 JSON>

JSON은 JavaScript Object Notation의 약자로, XML과 같이 텍스트를 기반으로 한 데이터 교환 표준 포맷이다. ‘속성-값 쌍’ 또는 ‘키-값 쌍’으로 이루어진 데이터 오브젝트를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하며, 기계는 물론 사람이 읽기에도 편한 구조로 되어 있다. 또한 언어에 독립적이라는 특징을 갖고 있어 C++, C#, JAVA, Perl등 다양한 언어와 플랫폼에 무관하게 사용할 수 있다. 즉 다양한 프로그래밍 언어에 의해 parsing 될 수 있으며, HttpRequest 객체를 사용해 서버로부터 데이터를 전송 받을 수도 있다. XML에 비해 데이터 저장에 필요로 하는 사이즈가 작은 등 다양한 이점을 가지고 있어 최근 데이터 저장 등에 많이 사용되는 편이다.

### 8.3 Protocol Description

#### A. Overview

본 항목에서는 DOCKING에서 사용되는 메시지들에 대해 JSON에 대응하는 Attribute와 Value로 나누어 작성한다. HTTP(HyperText Transfer Protocol)에서 전송되는 메시지는 request와 response로 구분된다. request는 client에서 server로

전송하는 메시지이고, response는 server에서 client로 전송하는 메시지를 말한다. 본 항목에서는 각 프로토콜에 대해 request와 response를 나누어 서술한다. 또한 이 장의 테이블들에 대해서는 캡션을 생략한다.

## B. Sign up with Naver Account Protocol

### a. Request

Attribute	Value	
user_data	Attribute	Value
	SSN	사용자의 고유식별번호
	Name	사용자의 이름
	Nickname	사용자의 닉네임
	age	사용자의 나이
	sex	사용자의 성별
Nprofile	사용자의 Naver profile	
linking	사용자의 Naver 연동 동의 버튼 클릭	

### b. Response

Attribute	Value
signup_verified	가입 성공 여부

## C. Sign up with Docking Account Protocol

### a. Request

Attribute	Value	
user_data	Attribute	Value
	SSN	사용자의 고유식별번호
	ID	사용자의 ID
	PW	사용자의 비밀번호
	Name	사용자의 이름
	Nickname	사용자의 닉네임
	age	사용자의 나이
	sex	사용자의 성별

### b. Response

Attribute	Value
signup_verified	가입 성공 여부

## D. Log in with Naver Account Protocol

### a. Request

Attribute	Value
Nlogin_button_clicked	사용자의 Naver 로그인 버튼 클릭

### b. Response

Attribute	Value
login_verified	로그인 성공 여부

## E. Log in with Docking Account Protocol

### a. Request

Attribute	Value	
user_data	Attribute	Value
	ID	사용자의 ID
	PW	사용자의 비밀번호

### b. Response

Attribute	Value
login_verified	로그인 성공 여부

## F. Main Page Display Protocol

### a. Request

Attribute	Value
login_success	사용자의 로그인 성공

### b. Response

Attribute	Value	
contents_data	Attribute	Value
	SSN	게시물의 고유번호
	substance	게시물의 사진이나 영상
	text	게시물의 글
	date	게시물의 작성날짜
	like	게시물의 좋아요 수

## G. Contents Upload / Edit Protocol

### a. Request

Attribute	Value	
contents_data	Attribute	Value
	SSN	게시물의 고유번호
	substance	게시물의 사진이나 영상
	text	게시물의 글
	date	게시물의 작성날짜
	like	게시물의 좋아요 수
account_SSN	게시물 작성자의 고유번호	
Post_or_Edit	첫 게시인지, 또는 기존 게시물 수정인지 표시	

### b. Response

Attribute	Value
postcontent_verified	게시물 포스팅 성공 여부

## H. Contents (include comments) Display Protocol

### a. Request

Attribute	Value
content_fragment_clicked	사용자의 게시물 클릭

### b. Response

Attribute	Value	
contents_data	Attribute	Value
	SSN	게시물의 고유번호
	substance	게시물의 사진이나 영상
	text	게시물의 글
	date	게시물의 작성날짜
	like	게시물의 좋아요 수
account_SSN	게시물 작성자의 고유번호	
comments	게시글에 달린 댓글들 리스트	
comments_data	Attribute	Value
	C_Num	(코멘트)댓글의 고유번호
	substance	댓글의 사진이나 영상
	date	댓글의 글
	like	댓글의 작성날짜
	parent_content	해당 댓글이 달려 있는 게시물 SSN

## I. Comments Upload / Edit Protocol

### a. Request

Attribute	Value	
comments_data	Attribute	Value
	C_Num	(코멘트)댓글의 고유번호
	substance	댓글의 사진이나 영상
	date	댓글의 글
	like	댓글의 작성날짜
	parent_content	해당 댓글이 달려 있는 게시물 SSN
account_SSN	댓글 작성자의 고유번호	
Post_or_Edit	첫 게시인지, 또는 기존 댓글 수정인지 표시	

### b. Response

Attribute	Value
postcomment_verified	댓글 포스팅 성공 여부

## J. Follow Protocol

### a. Request

Attribute	Value
-----------	-------

followee_SSN	팔로우 요청을 받은 사용자의 고유식별번호
follower_SSN	팔로우 요청한 사용자의 고유식별번호

b. Response

Attribute	Value
follow_verified	팔로우 성공 여부

## K. Contents Search Protocol

a. Request

Attribute	Value
search_filter	검색 필터 목록
search_word	검색 요청한 문자열

b. Response

Attribute	Value
contents_data	Attribute
	Value
	SSN
	substance
	text
	date
account_SSN	like
comments	게시물 작성자의 고유번호
	게시글에 달린 댓글들 리스트

## L. Walking Information Save Protocol

a. Request

Attribute	Value
walking_data	Attribute
	Value
	Dog_SSN
	start_time
	end_time
	elapse_time
account_SSN	distance
walking_history	산책한 사용자의 고유번호
	산책한 사용자의 산책 히스토리

b. Response

Attribute	Value
saveWalking_verified	산책 정보 저장 성공 여부

## M. Walking History Display Protocol

a. Request



Attribute	Value
walking_history_button_clicked	사용자의 walking history 조회 버튼 클릭
account_SSN	사용자의 고유번호

b. Response

Attribute	Value	
walking_data	Attribute	Value
	Dog_SSN	반려견의 고유번호
	start_time	산책 시작 시간
	end_time	산책 종료 시간
	elapse_time	산책 지속 시간
	distance	산책 거리

## N. Walking Mate Recommend Protocol

a. Request

Attribute	Value	
user_data	Attribute	Value
	SSN	사용자의 고유식별번호
	ID	사용자의 ID
	PW	사용자의 비밀번호
	Name	사용자의 이름
	Nickname	사용자의 닉네임
	age	사용자의 나이
	sex	사용자의 성별
pet_data	Attribute	Value
	SSN	반려견의 고유식별번호
	Name	반려견의 이름
	age	반려견의 나이
	weight	반려견의 몸무게
	sex	반려견의 성별
	species	반려견의 견종
otherUser_data	추천가능한 상태의 다른 사용자들의 데이터	
otherPet_data	추천가능한 상태의 다른 반려견들의 데이터	
recommend_criteria	산책메이트 추천 기준	

b. Response

Attribute	Value
recommend_rate	추천 점수
recommended_userSSN	추천된 다른 유저의 고유식별번호

## O. Start Direct Message Protocol

a. Request

Attribute	Value
-----------	-------

user_SSN	사용자의 고유식별번호
mate_SSN	매칭된 사용자의 고유식별번호

b. Response

Attribute	Value
startDM_verified	다이렉트 메시지 기능이 시작되었는지 여부

## P. Smartband Pairing Protocol

a. Request

Attribute	Value
band_MAC	밴드의 MAC주소
pet_SSN	반려견의 고유식별번호
account_SSN	사용자의 고유식별번호

b. Response

Attribute	Value
pairing_verified	페어링 성공 및 저장되었는지 여부

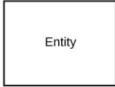
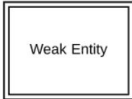
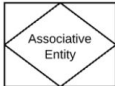
## 9 Database Design

### 9.1 Objectives


Database Design은 요구사항 명세서에서 작성한 데이터베이스 요구사항을 바탕으로 하여 수정 및 변경사항을 반영하여 작성되었다. 요구사항을 바탕으로 ER Diagram을 작성하고, 이를 이용하여 Relational Schema를 작성한 뒤, Normalization을 통해 Redundancy와 Anomaly를 제거한 후 마지막으로 SQL DDL을 작성한다.

### 9.2 Entity-Relationship Model

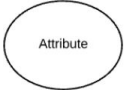


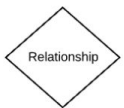
Entity는 분리된 물체 하나를 표현한다. Entity는 사각형으로 표현되며, Relationship은 다이아몬드로 표현된다. Entity나 Relationship은 Attributes를 가질 수 있으며, 이 Attributes는 관계 집합에 실선으로 연결된 타원형으로 표현한다. Weak Entity는 다른 Entity와의 연결 관계를 통해서만 해당 Entity의 객체들이 unique하게 구별될 수 있는 Entity로서, 두 줄로 된 사각형으로 표현된다.

Entity Symbol	Name	Description
	Strong entity	These shapes are independent from other entities, and are often called parent entities, since they will often have weak entities that depend on them. They will also have a primary key, distinguishing each occurrence of the entity.
	Weak entity	Weak entities depend on some other entity type. They don't have primary keys, and have no meaning in the diagram without their parent entity.
	Associative entity	Associative entities relate the instances of several entity types. They also contain attributes specific to the relationship between those entity instances.

<Figure 9.1 Entity Symbol>

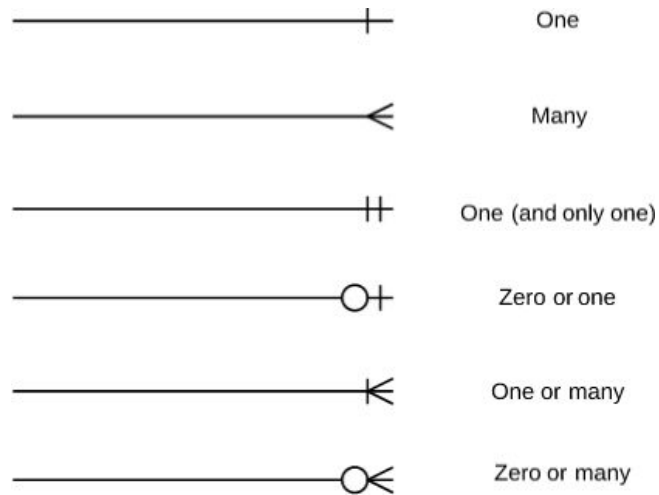
Relationship Symbol	Name	Description
	Relationship	Relationships are associations between or among entities.
	Weak relationship	Weak Relationships are connections between a weak entity and its owner.

<Figure 9.2 Relationship Symbol>

Attribute Symbol	Name	Description
	Attribute	Attributes are characteristics of an entity, a many-to-many relationship, or a one-to-one relationship.
	Multivalued attribute	Multivalued attributes are those that can take on more than one value.
	Derived attribute	Derived attributes are attributes whose value can be calculated from related attribute values.
	Relationship	Relationships are associations between or among entities.

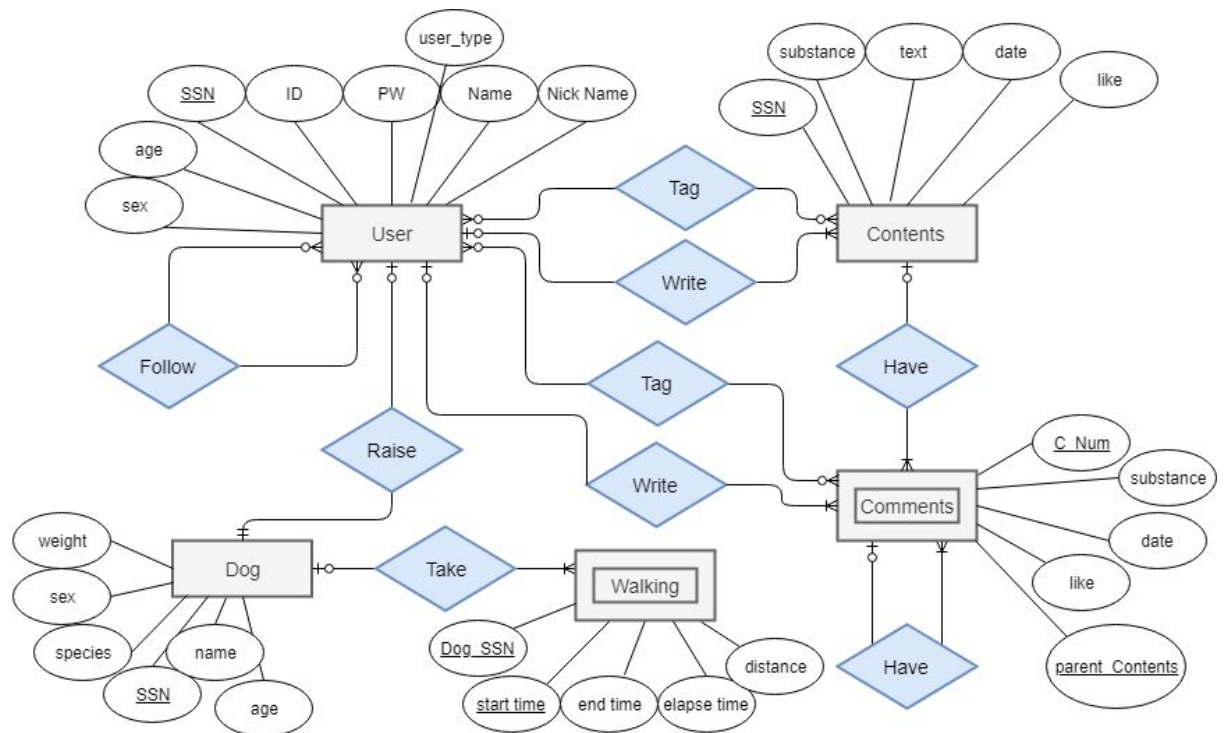
<Figure 9.3 Attribute Symbol>

Relationship은 두 개 이상의 Entity들의 연관 관계를 표현한다. 모든 Entity는 고유하게 식별되는 Attributes 집합을 가지고 있어야 하며, 최소한의 고유 식별 Attributes의 집합은 Entity의 기본 키(Primary key)로서, 해당 Entity의 객체들을 unique하게 구별해주는 역할을 수행한다. Entity와 Entity, Relationship 간 화살표들의 의미는 다음과 같다.



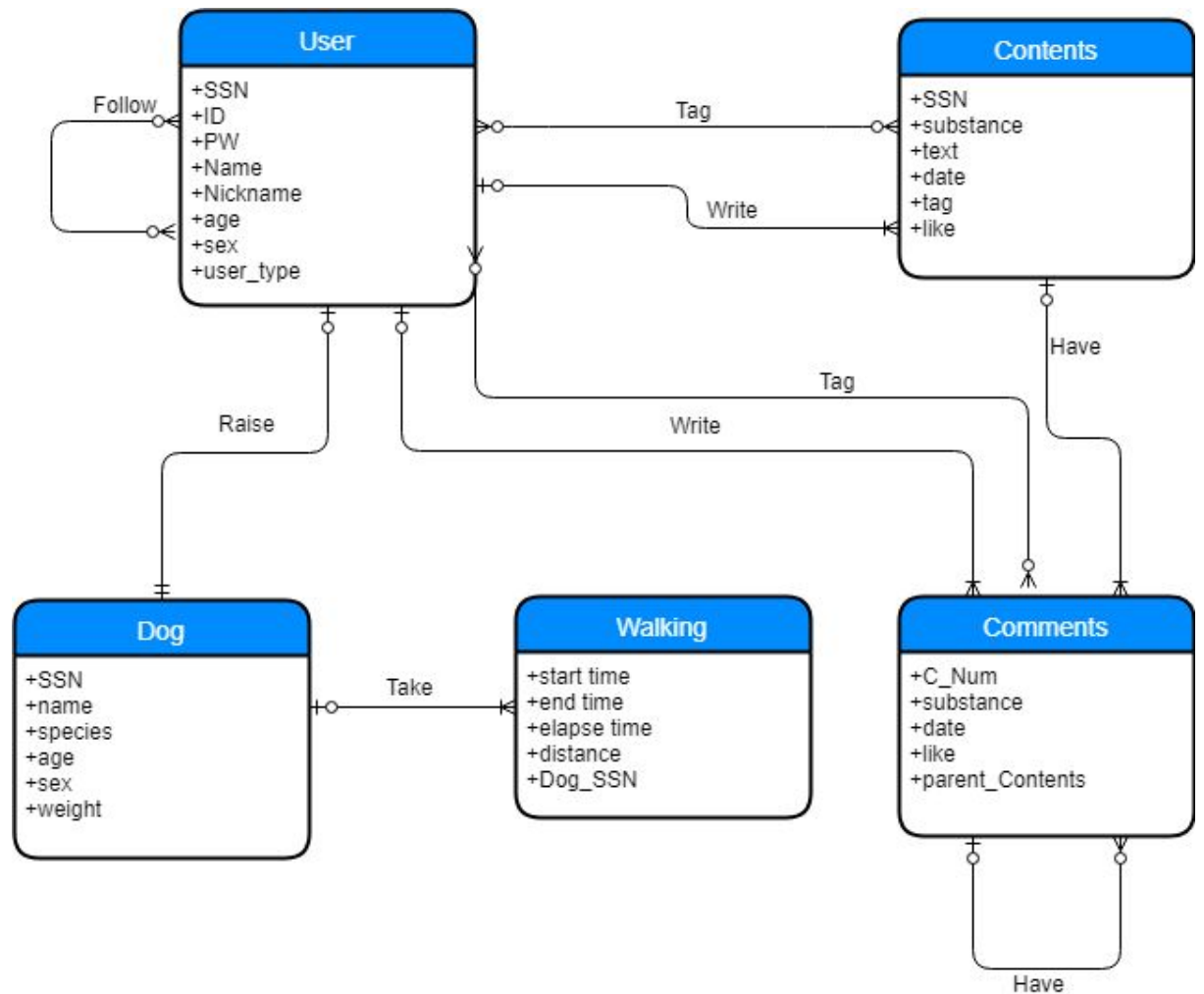
〈Figure 9.4 Cardinality and ordinality of ER Diagram〉<sup>1</sup>

Docking의 전체적인 ER Diagram은 다음과 같다.



〈Diagram 9.1 Overall ER Diagram ver1〉

<sup>1</sup> <https://www.lucidchart.com/pages/ER-diagram-symbols-and-meaning>

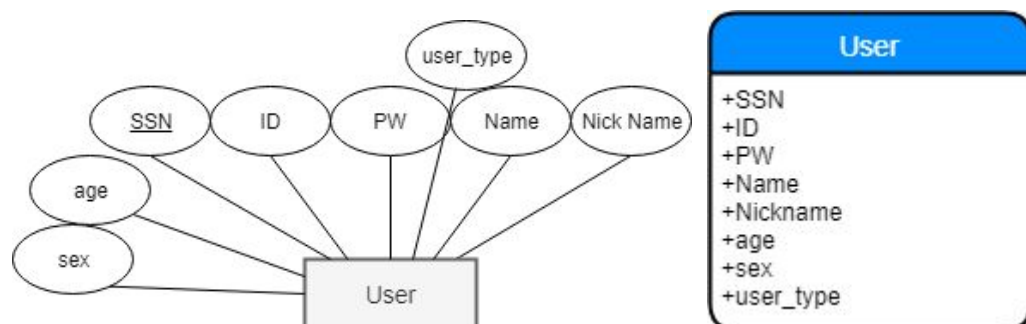


<Diagram 9.2 Overall ER Diagram ver2><sup>2</sup>

### 9.2.1 Entities

**Primal key**는 밑줄과 볼드체로 표기하고, *optional 한 attributes*는 이탤릭체로 표기한다. 이탤릭체로 표기한 것 이외의 attributes는 entity의 객체를 만들 때 반드시 기입되어야 한다.

#### A. User

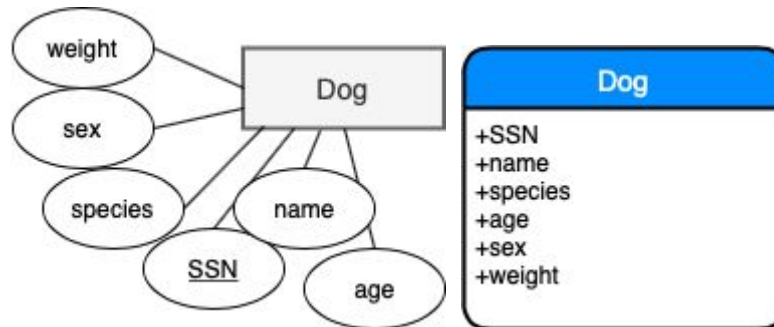


<Diagram 9.3 User Entity Diagram>

<sup>2</sup> 가독성과 정확한 정보기술을 위해 두 가지 형태의 ER diagram을 함께 사용한다. 표현하고 있는 내용은 같다.

User Entity는 Docking을 이용하는 사용자에게 대한 데이터로, SSN, ID, PW, Name, Nickname, user\_type, age, sex의 attributes가 있다.

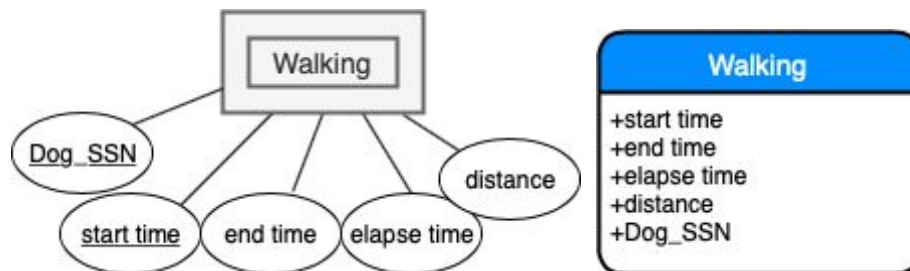
#### B. Dog



<Diagram 9.4 Dog Entity Diagram>

Dog Entity는 Docking을 이용하는 반려인들의 반려견들을 위한 데이터로, SSN, name, species, age, sex, weight의 attributes가 있다. 검색의 이점을 위해 Dog SSN은 반려인의 User SSN과 같다.

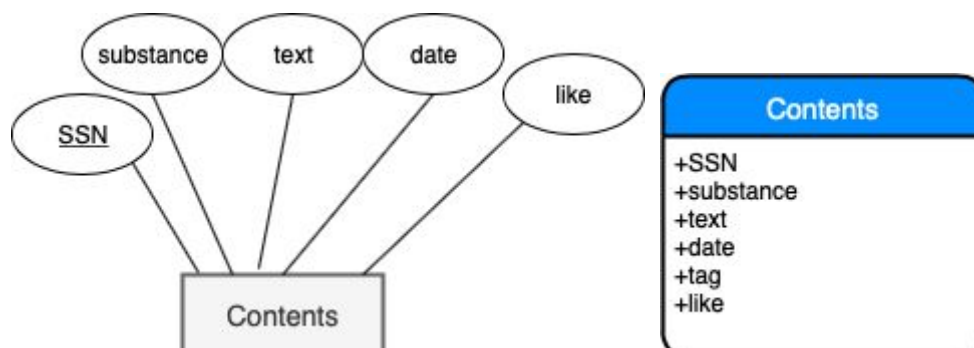
#### C. Walking



<Diagram 9.5 Walking Entity Diagram>

Walking Entity는 반려견들의 산책 데이터로, Dog\_SSN, start time, end time, elapse time, distance의 attributes가 있다. 해당 Entity는 Weak Entity로, Dog Entity에서 온 Dog\_SSN과 start time을 묶어 PK로서 사용한다.

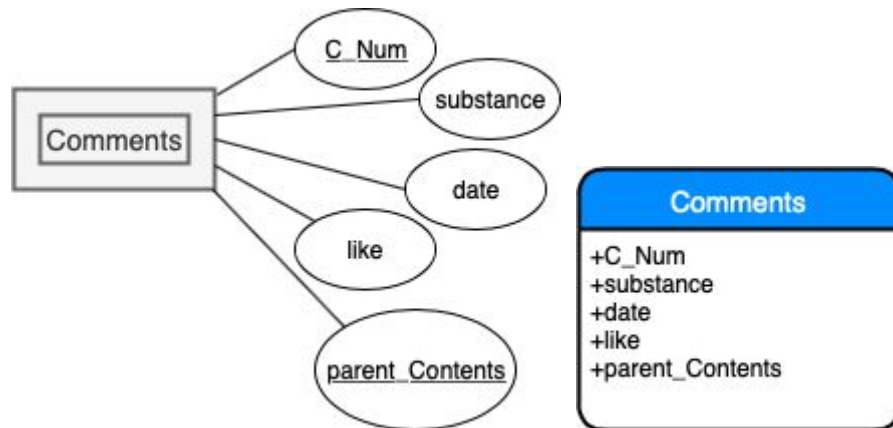
#### D. Contents



<Diagram 9.6 Contents Entity Diagram>

Contents Entity는 사용자들이 게시하는 게시물들에 관한 데이터로, SSN, substance, text, date, like의 attributes가 있다.

#### E. Comments

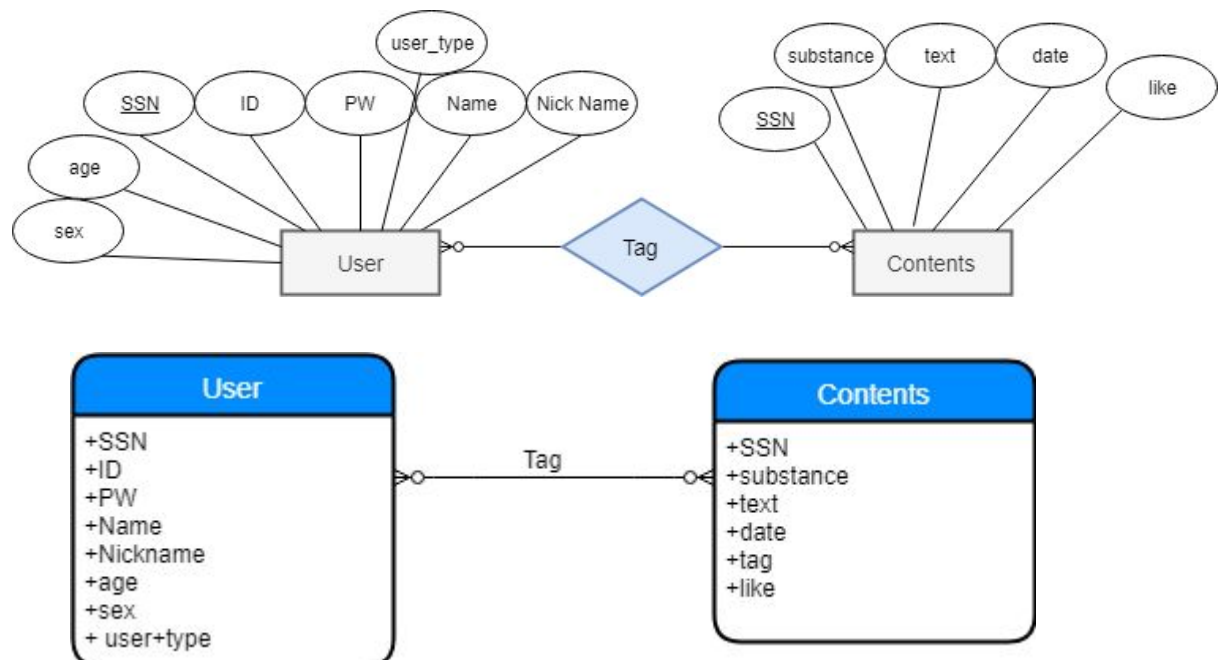


<Diagram 9.7 Comments Entity Diagram>

Comments Entity는 사용자들이 게시하는 댓글에 관한 데이터로, C\_Num, parent\_Contents, substance, date, like의 attributes가 있다. 해당 Entity는 weak entity로, C\_Num과 Contents의 SSN인 parent\_Contents를 결합하여 PK로 사용한다.

## 9.2.2 Relationships

#### A. Tag; Contents

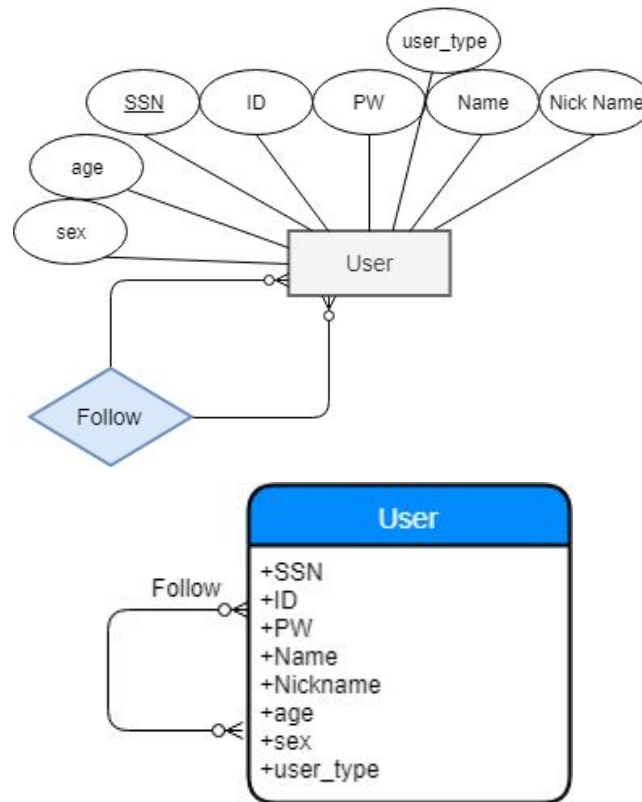




<Diagram 9.8 Tag-Contents Relation Diagram>

한 명의 유저가 여러 개의 콘텐츠에 태그될 수 있으며, 한 콘텐츠는 여러 명의 유저를 태그할 수 있다. 콘텐츠는 유저를 태그하지 않을 수 있으며, 유저는 콘텐츠에 태그되지 않을 수 있다. (many to many, optional to optional)

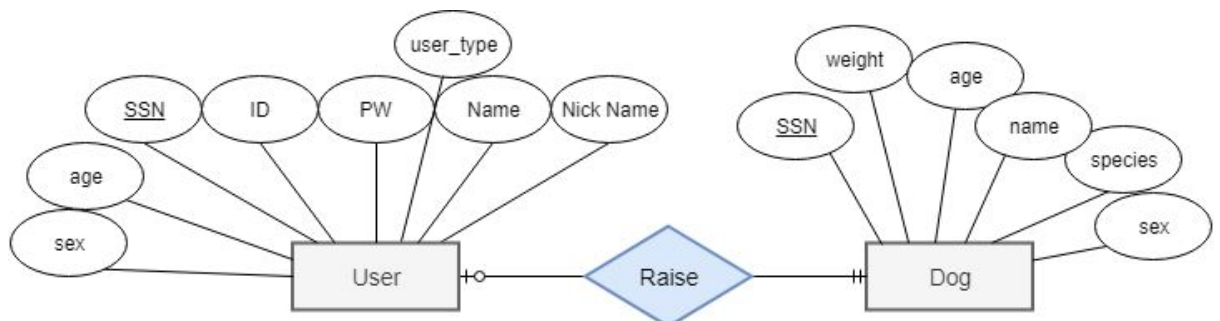
## B. Follow

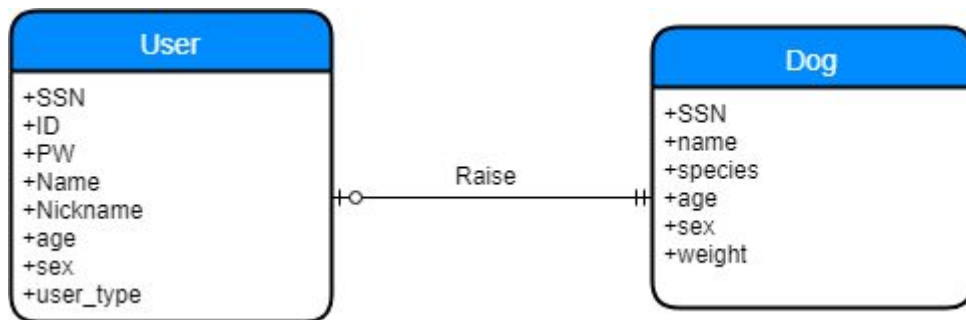


<Diagram 9.9 Follow Relation Diagram>

유저가 다른 유저를 팔로우할 수 있다. 한 명의 유저가 여러 명의 유저를 팔로우할 수 있으며, 한 명의 유저가 여러 명의 유저들로부터 팔로우될 수 있다. 어떤 유저는 팔로우하지 않을 수 있으며, 어떤 유저는 팔로우되지 않을 수 있다. (many to many, optional to optional)

## C. Raise

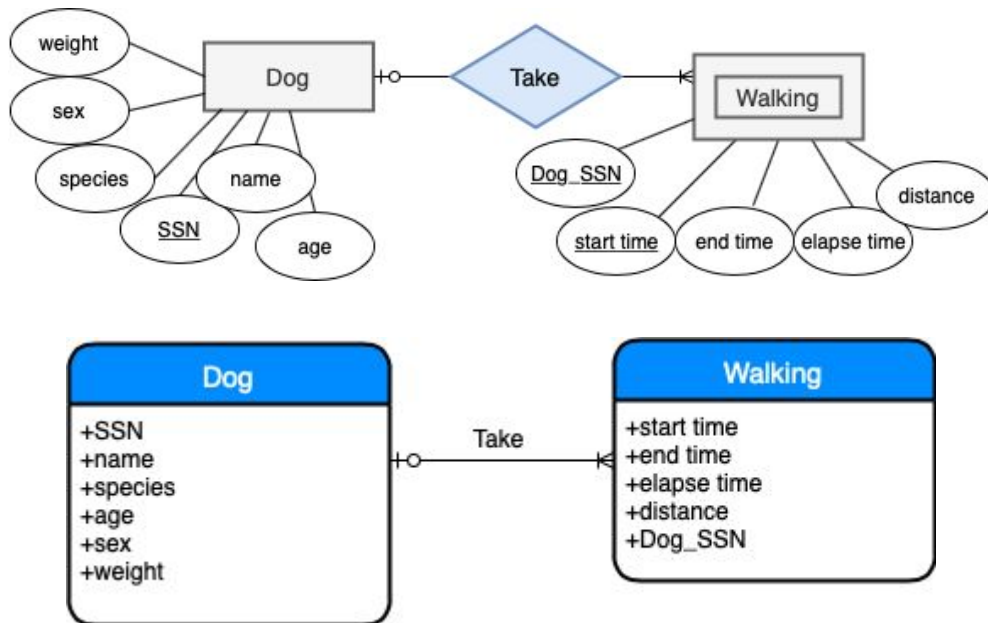




<Diagram 9.10 Raise Relation Diagram>

한 명의 유저는 하나의 강아지를 반려견으로 등록할 수 있으며, 한 마리의 강아지는 반드시 그의 반려인으로서의 유저를 가지고 있어야 한다. (optional to mandatory, one to one)

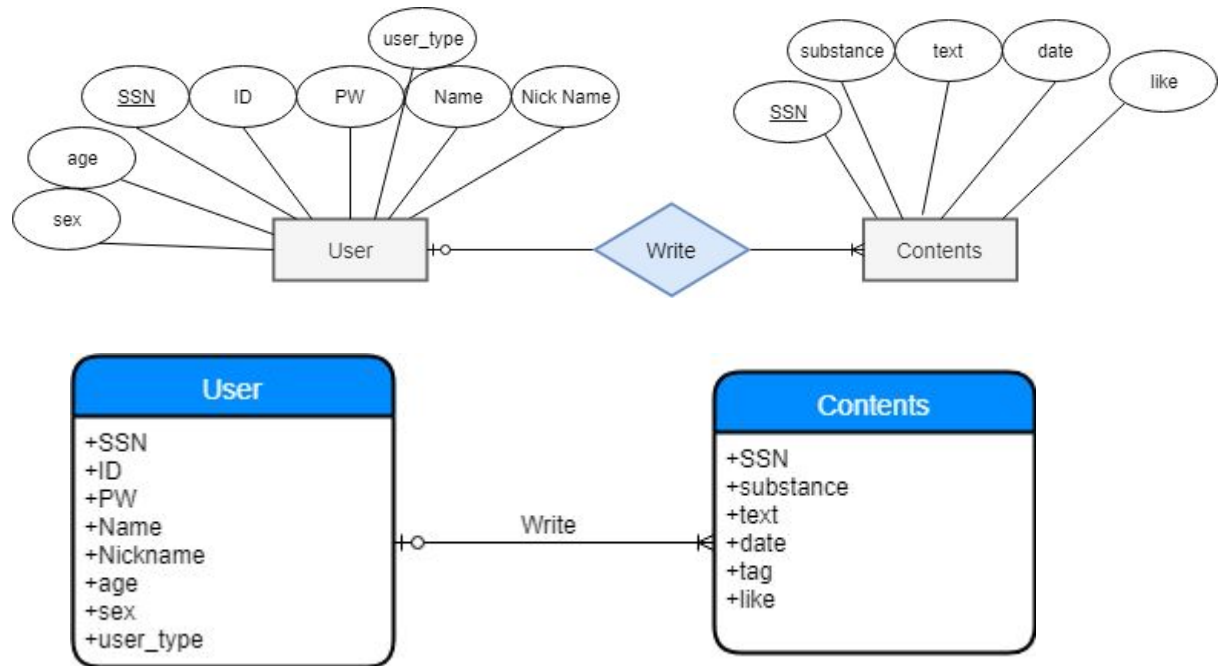
#### D. Take



<Diagram 9.11 Take Relation Diagram>

한 마리의 강아지는 여러 개의 산책 기록을 가질 수 있으며, 하나의 산책은 한 마리의 강아지와 연결된다. 하나의 산책은 반드시 그 산책을 진행하였던 강아지 정보를 가지고 있어야 한다. 어떤 강아지는 산책 기록이 없을 수도 있다. (optional to mandatory, one to many)

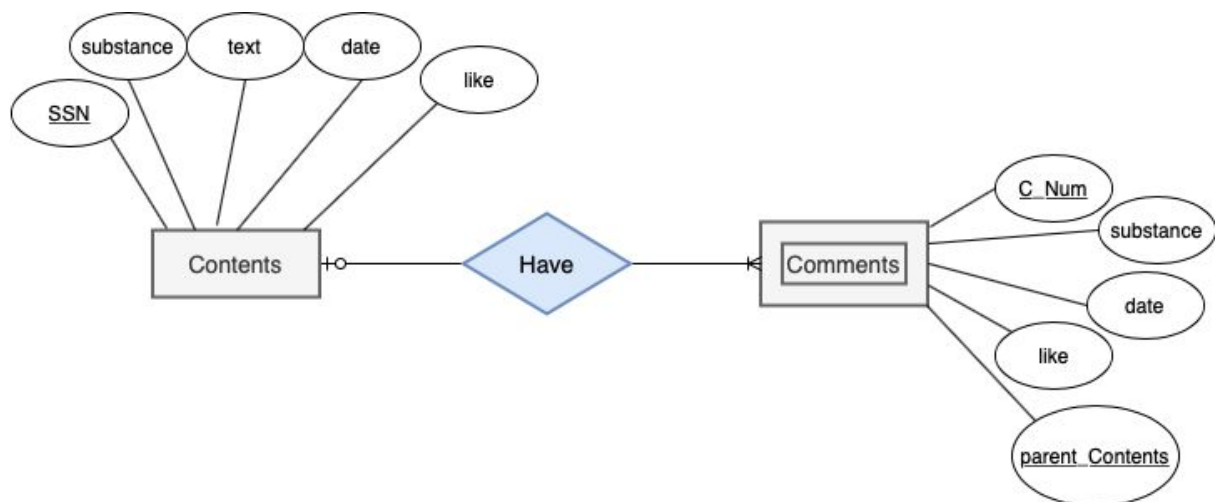
### E. Write; Contents

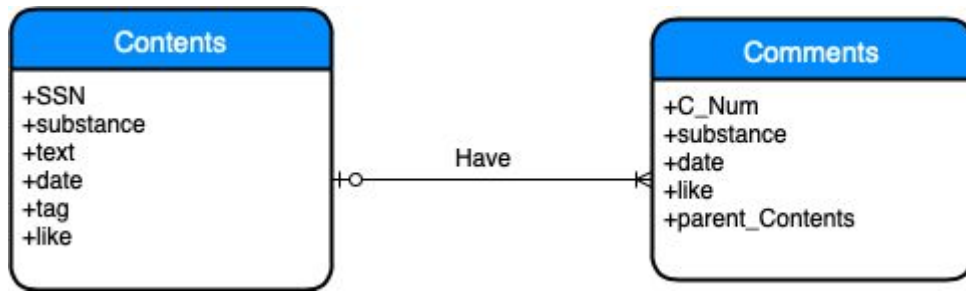


<Diagram 9.12 Write-contents Relation Diagram>

한 명의 유저는 여러 개의 콘텐츠를 쓸 수 있으며, 하나의 콘텐츠는 한 명의 글쓴이를 반드시 가진다. 어떤 유저는 콘텐츠를 쓰지 않을 수도 있다. (optional mandatory, one to many)

### F. Have; Reply

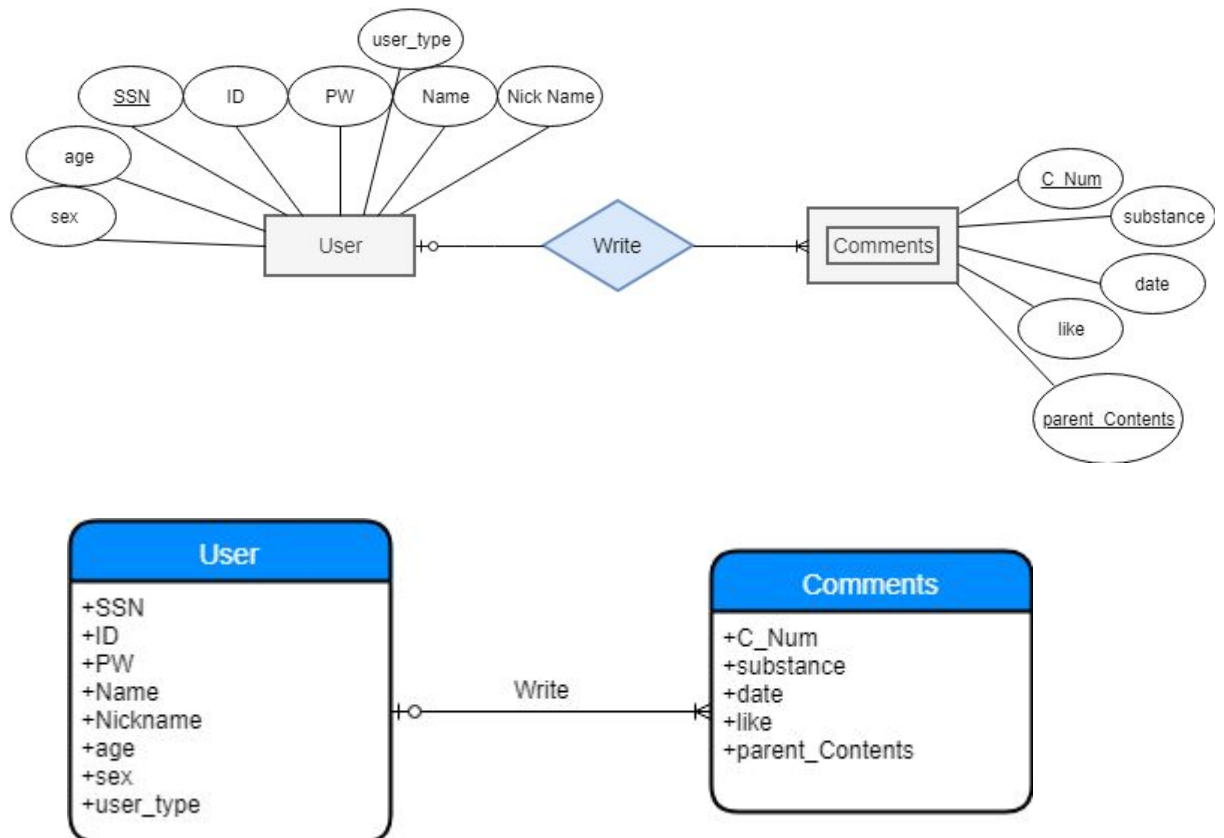




<Diagram 9.13 Have-reply Relation Diagram>

하나의 콘텐츠는 여러 개의 댓글을 가질 수 있으며, 하나의 댓글은 하나의 콘텐츠에만 귀속된다. 어떤 콘텐츠는 댓글을 가지지 않을 수도 있지만, 모든 댓글은 연관된 하나의 콘텐츠가 존재한다. (optional to mandatory, one to many)

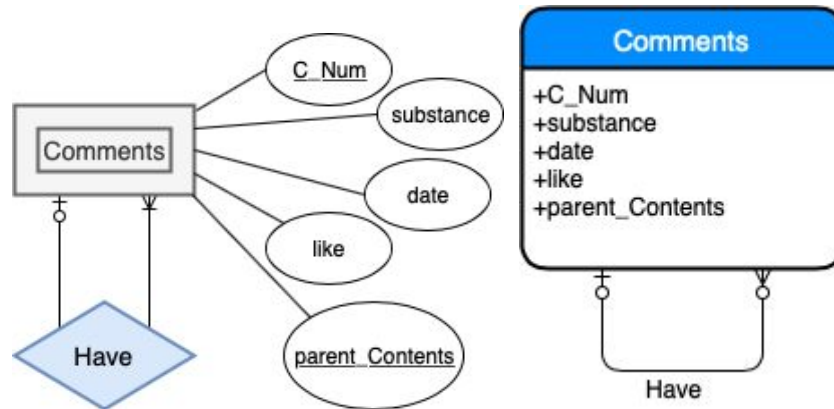
#### G. Write; Comments



<Diagram 9.14 Write-comments Relation Diagram>

한 명의 유저는 여러 개의 댓글을 쓸 수 있다. 하나의 댓글은 한 명의, 그 댓글을 쓴 유저를 가진다. 어떤 유저는 댓글을 쓰지 않을 수도 있다. 모든 댓글은 그 댓글을 쓴 사용자를 가진다. (optional to mandatory, one to many)

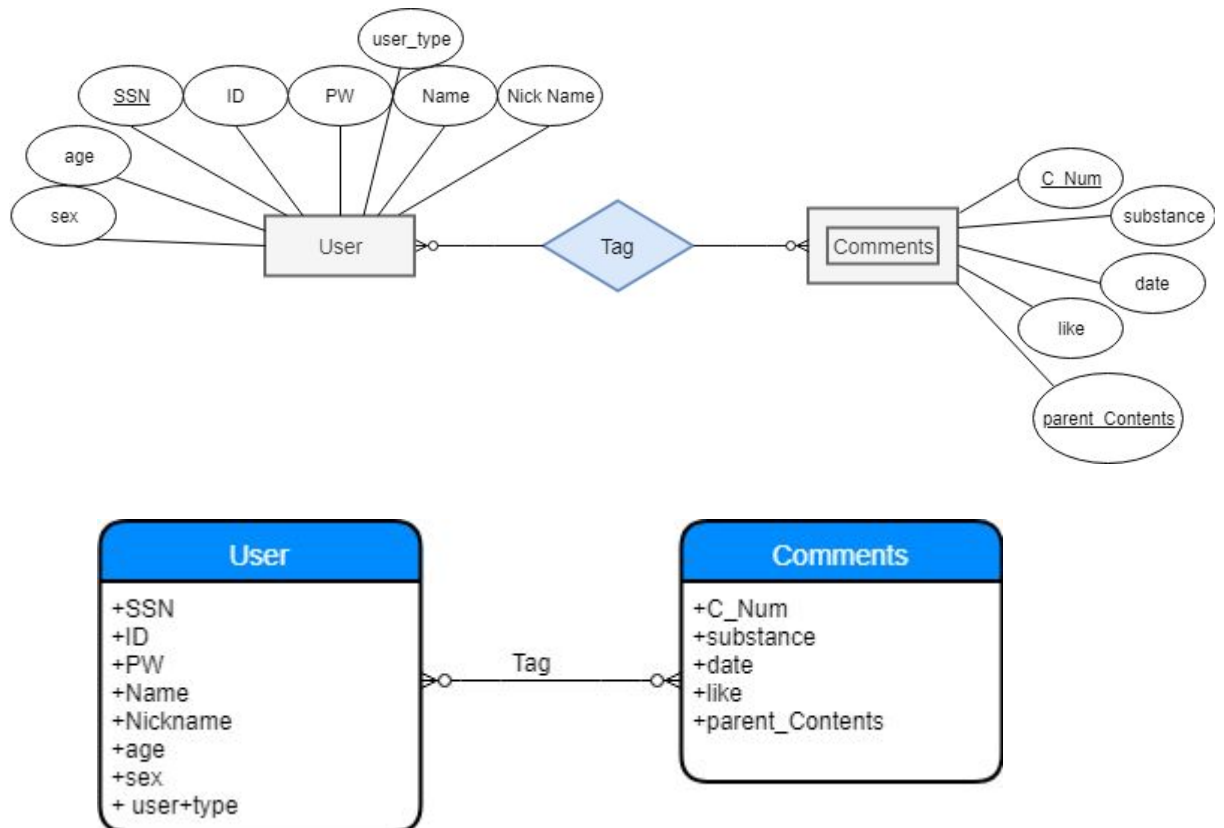
## H. Have; Re-reply



<Diagram 9.15 Have-re-reply Relation Diagram>

한 콘텐츠가 있고 거기에 댓글이 달려 있을 때, 해당 댓글에 댓글을 다는 식의 대댓글을 한 번만 허용한다. (대댓글에 댓글을 달 수는 없다.) 한 댓글은 여러 개의 대댓글을 가질 수 있으며, 대댓글은 그것이 달려 있는 하나의 댓글을 가져야 한다. 어떤 댓글은 대댓글을 가지지 않을 수도 있지만, 대댓글은 댓글을 가져야 한다. (optional to mandatory, one to many)

## I. Tag; Comments



<Diagram 9.16 Tag-Comments Relation Diagram>

한 유저가 여러 개의 댓글에 태그될 수 있으며, 한 댓글이 여러 명의 유저를 태그할 수 있다. 어떤 유저는 댓글에 태그되지 않을 수 있으며, 어떤 댓글은 유저를 태그하지 않을 수 있다. (optional to optional, many to many)

## 9.3 Relational Schema

### A. User

User	
PK	SSN
	ID
	PW
	Name
	Nickname
	age
	sex
	user_type

<Table 9.1 Relational Schema of User>

#### (1) Description

User Entity에 관한 테이블이다. SSN, ID, PW, Name, Nickname, user\_type 속성에 NULL을 허용하지 않으며, age, sex에 NULL을 허용한다.

#### (2) Function Dependency (FD)

SSN → ID, PW, Name, Nickname, age, sex, user\_type

ID → PW, SSN, Nickname, age, sex, user\_type

### B. Dog

Dog	
PK	SSN
	name
	species
	age
	sex

	weight
--	--------

<Table 9.2 Relational Schema of Dog>

(1) Description

Dog Entity, Raise 관계를 표현하는 Table이다. PK인 SSN을 반려인 User Entity의 SSN과 같도록 설정함으로써 Raise관계를 표현한다. species, age, sex, weight에 NULL을 허용한다.

(2) Function Dependency (FD)

SSN → name, species, age, sex, weight

## C. Walking

Walking	
PK	start time
	end time
	elapse time
	distance
FK, PK	Dog_SSN

<Table 9.3 Relational Schema of Walking>

(1) Description

Take 관계를 표현하는 테이블이다. Dog\_SSN을 Foreign key로 받아옴으로써 Take 관계를 표현한다. elapse time과 distance는 NULL을 허용한다.

(2) Function Dependency (FD)

{Dog\_SSN, start time} → end time, elapse time, distance

{start time, end time} → elapse time

## D. Contents

Contents	
PK	SSN
	substance
	text
	date
	tag
	like

FK	User_SSN
----	----------

<Table 9.4 Relational Schema of Contents>

(1) Description

Contents Entity와 Write; Contents 관계를 표현하는 테이블이다. User\_SSN을 Foreign key로 받아옴으로써 Write; Contents 관계를 표현한다. text에 NULL을 허용한다.

(2) Function Dependency (FD)

SSN  $\rightarrow$  {substance, text, date, tag, like, User\_SSN}

## E. Comments

Comments	
PK	C_Num
	substance
	date
	like
	tag
FK, PK	Contents_SSN
FK	User_SSN
FK	super C_Num

<Table 9.5 Relational Schema of Comments>

(1) Description

Comments Entity와 Write; Comments 관계, Have; Reply 관계, Have; Re-reply 관계를 표현하는 Table이다. User\_SSN을 Foreign key로 받아옴으로써 Write; Comments 관계를 표현하고, Contents\_SSN을 FK로 받아옴으로써 Write; Comments 관계를 표현하고, super C\_Num이라는 이름으로 C\_Num을 Foreign key를 받아옴으로써 Have; Re-reply 관계를 표현한다. 모든 속성에 NULL을 허용하지 않는다.

(2) Function Dependency (FD)

{C\_Num, Contents\_SSN}  $\rightarrow$  substance, date, like, tag, User\_SSN, super C\_Num

## F. Tag

Tag
-----



FK, PK	Contents_SSN
FK, PK	User_SSN
	horizontal location
	vertical location
	contents_type

<Table 9.6 Relational Schema of Tag>

(1) Description

Tag; Contents 관계와 Tag; Comments 관계를 표현하는 Table이다. Contents\_SSN과 User\_SSN을 Foreign key로 받아옴으로써 두 관계를 표현하고, contents\_type에 Tag가 달린 대상이 Contents인지 Comments인지를 표시해줌으로써 두 관계를 구분해준다. horizontal location과 vertical location 속성에 NULL이 허용된다.

(2) Function Dependency (FD)

{Contents\_SSN, User\_SSN} → horizontal location, vertical location, contents\_type

## G. Follow

Follow	
FK, PK	following User_SSN
FK, PK	followed User_SSN
	date

<Table 9.7 Relational Schema of Follow>

(1) Description

Follow 관계를 표현해주는 Table이다. following User\_SSN, followed User\_SSN을 Foreign key로 받아옴으로써 Follow 관계를 표현한다. 모든 속성에 NULL이 허용되지 않는다.

(2) Function Dependency (FD)

{following User\_SSN, followed User\_SSN} → date

## 9.4 SQL DDL

### A. User

data definition language form of the table User

```

TABLE USER(
SSN int PRIMARY KEY NOT NULL,
ID VARCHAR(100) NOT NULL,
PW VARCHAR(50) NOT NULL,
Nickname VARCHAR(100) NOT NULL,
age int ,
sex CHAR(1),
user_type int)

CREATE DOMAIN SEX CHAR(1)(
DEFAULT NULL
CONSTRAINT VALD-SEX CHECK (VALUE IN('남', '여')));

```

〈Table 9.8 data definition language form of the table User〉

## B. Dog

data definition language form of the table Dog

```

TABLE DOG(
SSN int PRIMARY KEY NOT NULL,
name VARCHAR(100) NOT NULL,
species VARCHAR(100),
age int,
sex VARCHAR(4),
weight int)

CREATE DOMAIN SEX CHAR(1)(
DEFAULT NULL
CONSTRAINT VALD-SEX CHECK (VALUE IN('남', '여')));

```

〈Table 9.9 data definition language form of the table Dog〉

## C. Walking

data definition language form of the table Walking

```

TABLE WALKING(
starttime DATETIME NOT NULL,
endtime DATETIME NOT NULL,
elapsetime int,
distance int,
Dog_SSN int,
PRIMARY KEY(Dog_SSN, starttime),
CONSTRAINT Dog_SSN FOREIGN KEY(Dog_SSN) REFERENCES USER (SSN)
ON DELETE CASCADE ON UPDATE CASCADE)

```

〈Table 9.10 data definition language form of the table Walking〉

## D. Contents

data definition language form of the table Contents

```
TABLE CONTENTS(  
SSN int PRIMARY KEY NOT NULL,  
substance LONGTEXT NOT NULL,  
text LONGTEXT,  
date DATETIME NOT NULL,  
tag INT(1) DEFAULT 0,  
like int NOT NULL,  
User_SSN int NOT NULL,  
CONSTRAINT User_SSN FOREIGN KEY(User_SSN) REFERENCES USER (SSN)  
ON DELETE CASCADE ON UPDATE CASCADE)
```

<Table 9.11 data definition language form of the table Contents>

## E. Comments

data definition language form of the table Comments

```
TABLE COMMENTS(  
C_Num int NOT NULL,  
substance LONGTEXT NOT NULL,  
date DATETIME NOT NULL,  
like int NOT NULL,  
parent_Contents int NOT NULL,  
User_SSN int NOT NULL,  
Super_C_Num int DEFAULT NULL,  
PRIMARY KEY(parent_Contents, C_Num),  
CONSTRAINT User_SSN FOREIGN KEY(User_SSN) REFERENCES USER (SSN)  
ON DELETE CASCADE ON UPDATE CASCADE,  
CONSTRAINT parent_Contents FOREIGN KEY(parent_Contents) REFERENCES  
CONTENTS (SSN) ON DELETE CASCADE ON UPDATE CASCADE,  
CONSTRAINT super_C_Num FOREIGN KEY(Comments) REFERENCES  
COMMENTS (C_Num) ON DELETE CASCADE ON UPDATE CASCADE)
```

<Table 9.12 data definition language form of the table Comments>

## F. Tag

data definition language form of the table Tag

```
TABLE TAG(  
Contents_SSN int NOT NULL,  
User_SSN int NOT NULL,  
horizontal_lacation int DEFAULT NULL,  
vertical location int DEFAULT NULL,  
contents_type VARCHAR(10),  
PRIMARY KEY(Contents SSN, User_SSN),
```

```

CONSTRAINT User_SSN FOREIGN KEY(User_SSN) REFERENCES USER (SSN)
ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT Contents_SSN FOREIGN KEY(Contents SSN) REFERENCES
CONTENTS (SSN) ON DELETE CASCADE ON UPDATE CASCADE)

```

<Table 9.13 data definition language form of the table Tag>

## G. Follow

data definition language form of the table Follow

```

TABLE FOLLOW(
(following int NOT NULL,
follower int NOT NULL,
date DATE NOT NULL,
PRIMARY KEY(following, follower),
CONSTRAINT foollowing FOREIGN KEY(following) REFERENCES USER (SSN)
ON DELETE CASCADE ON UPDATE CASCADE,
CONSTRAINT follower FOREIGN KEY(follower) REFERENCES USER (SSN)
ON DELETE CASCADE ON UPDATE CASCADE)

```

<Table 9.14 data definition language form of the table Follow>

# 10 Testing plan

## 10.1 Objective

시스템이 요구사항에서 의도한 대로 작동하는것을 확인하고, 그 외의 결함을 찾기 위한 Test를 진행한다. Testing plan에서는 Testing policy와 Test case를 정의한다.

## 10.2 Testing Policy

Test는 Developing testing, release testing, user testing의 3 단계로 진행한다.

### 1) Developing testing

Developing test에서는 각 component와 subsystem의 기능을 검증한다.

### 2) Release testing

Release test에서는 요구사항 명세서의 요구사항이 반영되었음을 검증한다.

### 3) User testing

User Test에서는 실제 사용자의 환경에서 Test가 이루어진다.

## 10.3 Test case

### A. User information System

#### a. Sign up with Naver

1) User: Naver계정 연동을 통해 가입을 시도한다.

2) 시스템 : User DB를 통해 해당 Naver 계정의 존재 유무를 확인한다.

ㄱ) 성공

- 시스템: 가입 성공 메시지를 표시한다.  
DB에 user계정 정보가 추가된다.
- ㄴ) 실패  
시스템: 가입 실패 메시지를 표시한다.
- b. Sign up with Docking
- 1) User: 직접 ID와 개인정보를 입력후 가입 시도를 한다.
  - 2) 시스템: User DB를 통해 중복된 사용자를 확인한다.
    - ㄱ) 성공:
 

시스템: 가입 성공 메시지를 표시한다.  
DB에 user계정 정보가 추가된다.
    - ㄴ) 실패:
 

시스템: 가입 실패 메시지를 표시한다.
- c. Login with Naver
- 1) User: Naver계정 연동을 통해 가입을 시도한다.
  - 2) 시스템 : User DB를 통해 해당 Naver 계정의 존재 유무를 확인한다.
    - ㄱ) 성공:
 

시스템: 로그인 성공 메시지를 표시한다.  
Mainpage를 표시한다.
    - ㄴ) 실패:
 

시스템: 로그인 실패 메시지를 표시한다.
- d. Login With Docking
- 1) User: ID와 Password를 입력하고 로그인 시도를 한다.
  - 2) 시스템 : User DB를 통해 해당 계정의 존재 유무를 확인한다.  
계정이 존재할시 Password 일치 여부를 검사한다.
    - ㄱ) 성공:
 

시스템: 로그인 성공 메시지를 표시한다.  
Mainpage를 표시한다.
    - ㄴ) 실패:
 

시스템: 로그인 실패 메시지를 표시한다.
- e. Logout
- 1) User: 로그아웃 버튼을 누른다.
  - 2) 시스템: 로그아웃 확인 메시지를 표시한다.
  - 3) User: 로그아웃 확인창을 조작한다.
    - a) 로그아웃 확인버튼 누름  
로그아웃이 정상적으로 이루어지고 로그아웃 상태가 되며, 메인페이지로 돌아간다.
    - b) 로그아웃 취소버튼 누름  
로그아웃이 취소되며 확인창이 없어지고 기존 창으로 돌아간다.
- f. Mypage
- 1)
- g. Sign out
- 1) user: Sign out 버튼을 누른다
  - 2) 시스템:
    - ㄱ)Naver계정 사용자:
 

Sign out 확인 메시지를 표시한다.
    - ㄴ)Docking 계정 사용자:

Sign out 확인용으로 ID, Password를 받는다.

3) user:

ㄱ) Naver계정 사용자:

i) Sign out 버튼을 확인한다.

ii) Sign out을 취소한다.

ㄴ) Docking 계정 사용자:

i) Sign out 확인 ID/Password를 입력한다.

ii) Sign out을 취소한다.

4) 시스템:

ㄱ) Naver계정 사용자:

i) Sign out 성공 메시지를 표시한다, DB에서 사용자 계정을 탈퇴처리한다.

ii) Sign out취소메세지를 띄우고 원래 화면으로 돌아간다.

ㄴ) Docking 계정 사용자:

i) Sign out 확인 ID/Password를 확인한다.

i-i) ID/Password 일치시: 계정을 탈퇴처리한다.

i-ii) ID/Password 불일치시: 실패메세지를 표시한다.

i-iii) ID/Password 불일치 허용 횟수 초과시: APP을 잠근다.

ii) Sign out을 취소하고 원래 화면으로 돌아간다.

## B. Matching System

### a. Recommend match

1) user: Match 메뉴에 들어가 추천Matching을 신청한다.

2) 시스템: User 조건에 따라 자동으로 Match 상대를 선정한다.

3) user: 추천된 Match를 확인한다.

a) 추천된 Match를 선택한다.

b) 추천된 Match를 거부한다.

2) 시스템

a) 상대에게 Match 승낙 여부를 확인한다.

b) Recommend match로 돌아간다.

1) 상대 user:

a) Match를 승낙한다.

b) Match를 거부한다.

1) 시스템

a) 승낙시 Walking정보를 생성하고 DM을 연결한다.

b) Match 거부시 거부를 알리고 Recommend match로 돌아간다.

### b. Manual match

1) user: manual match 를 선택한다.

2) 시스템: Match 상대 List를 보인다

3) user: Manual match 상대를 선택한다.

4) 시스템: 선택된 상대에게 승낙확인 메세지를 보낸다.

5) 상대 user:

a) Match를 승낙한다.

b) Match를 거부한다.

6) 시스템

a) 승낙시 Walking정보를 생성하고 DM을 연결한다.

b) Match 거부시 거부를 알리고 Manual match로 돌아간다.

## C. Community System

- a. Reading
  - 1) user:post를 선택한다.
  - 2) 시스템: 선택된 Post의 정보를DB에서 불러와 표시한다.
- b. Writing
  - 1) user:Post를 작성한다.
  - 2) 시스템: 작성된 Post의 정보를 DB에 저장한다.
- c. editing
  - 1) user:Post edit을 선택한다.
  - 2) 시스템: Edit 창을 띄우고, 기존 Post의 내용을 표시한다.
  - 3) user: 수정후 Post 저장을 선택한다.
  - 4) 시스템: 수정된 Post 정보를 DB에 저장한다.
- d. Deleting
  - 1) user: Post Delete를 선택한다.
  - 2) 시스템: Delete 확인 선택창을 표시한다.
  - 3) user:
    - a) Post Delete 를 확인한다.
    - b) Post Delete 를 취소한다.
  - 4) 시스템:
    - a) Post를 삭제처리한다. Post 삭제 메시지를 표시한다.
    - b) Post 삭제를 취소하고 원래 화면으로 돌아간다.
- e. Deleting by manager
  - 1) 관리자: Post Delete를 선택한다.
  - 2) 시스템: Delete 확인 선택창을 표시한다.
  - 3) 관리자:
    - a) Post Delete 를 확인한다.
    - b) Post Delete 를 취소한다.
  - 4) 시스템:
    - a) Post를 삭제처리한다. Post 삭제 메시지를 표시한다.
    - b) Post 삭제를 취소하고 원래 화면으로 돌아간다.
- f. editing by manager
  - 1) 관리자 :Post edit을 선택한다.
  - 2) 시스템: Edit 창을 띄우고, 기존 Post의 내용을 표시한다.
  - 3) 관리자 : 수정후 Post 저장을 선택한다.
  - 4) 시스템: 수정된 Post 정보를 DB에 저장한다.
- D. Walking information
  - a. matched Schedule
    - 1) user:저장된 Schedule을 선택한다.
    - 2) 시스템: 저장된 Schedule 정보를 표시한다.
  - b. walking history
    - 1) user:저장된 History를 선택한다.
    - 2) 시스템: 저장된 History 정보를 표시한다.
- E. Band usage
  - a. Band sync
    - 1) user: Band sync를 선택한다.
    - 2) 시스템: Band Sync를 시도한다.
      - i) Sync 성공:

UserDB에 Band정보를 저장한다.

Sync 성공메세지를 표시한다.

ii) Sync 실패:

Sync 실패 메세지를 표시한다.

Sync초기화면으로 돌아간다.

1) User: Band 닉네임을 입력한다.

2) 시스템: Band 닉네임을 DB에 저장한다.

b. Band info

1) user: Band를 착용하고 산책을 한다.

2) 시스템: GPS 정보와 자이로정보(걸음수)를 수집한다.

3) user:산책을 종료한다

4) 시스템: GPS정보(산책경로)와 걸음수를 Walking history 에 저장한다.



# 11 Development Environment

## 11.1 Objectives

Development Environment에서는 개발자의 환경에 대해 설명한다. 사용하는 프로그래밍 언어와 IDE에 대해 서술한다.

## 11.2 Programming Language & IDE



<Figure 11.1 Android Studio Symbol>

Front end 앱 개발을 위해 Java 기반의 Android studio를 이용한다. Android Studio는 안드로이드 및 안드로이드 전용 어플 제작을 위한 공식 통합 개발 환경(IDE)다.



<Figure 11.2 Firebase Logo>

파이어 베이스는 구글의 모바일 및 웹 어플리케이션 개발 플랫폼이다. 본 프로젝트에서는 파이어베이스의 Realtime Database 서비스를 주로 이용할 예정이다. 이를 통해 서버를 따로 관리하지 않아도 되는 편리함을 얻을 수 있다.



<Figure 11.3 Arduino Logo>

아두이노의 통합 개발 환경(IDE)는 Java 와 C를 기반으로 개발되는 크로스 플랫폼 응용 소프트웨어다. 아두이노 개발환경은 C++을 사용하여 원하는 동작을 하도록 코딩을 하고 이것을 보드에 업로드하면 아두이노가 동작한다. 반려견 스마트밴드 구현을 위해 활용한다.

## 11.3 Version Management Tool



<Figure 11.4 Github Logo>

효율적인 개발과 버전 및 코드 관리, 공유를 위해 GitHub를 이용한다. GitHub는 전세계적으로 많이 사용되는 웹 기반의 호스팅 서비스로 신뢰도가 높은 서비스이다. 또한 오픈 소스가 많이 등록되어 있어, 이를 이용하는데 편리하고 유용한 서비스를 제공한다.

## 11.4 Coding Rule

Docking 시스템에서 사용한 Coding Rule은 다음과 같다.

1) Function 단위로 주석을 단다.

병행개발 후 통합을 위해, 다른 개발자가 각 부분의 기능, 원리에 대해 알아야한다. 따라서 주석을 추가하여 코드의 설명을 돕되 line-by-line 으로 주석을 작성하기에는 overload가 심하다. 따라서 Function 단위로 주석을 다는 것을 규칙으로 한다.

2) GitHub를 이용하는 경우, 두 명 이상이 동시에 같은 파일에 대해 수정, 작업하는 경우가 없도록 한다.

3) Class import를 미리 하지 않는다.

안드로이드 스튜디오에서 해당 코드에서 class 가 import 되지 않고 사용된다면 자동으로 class 가 import 된다. 따라서 꼭 사용되는 class 만 import 될 수 있는데, 미리 class 를 import 한다면 쓰지 않을 가능성이 있는 class까지 포함하는 무거운 프로그램을 생성 할 수 있다.

## 12 Development Plan

### 12.1 Objective

Development Plan에서는 개발 계획에 대해 서술한다. 이 때, Gantt chart를 이용하여 개발 계획과 실제 개발 흐름에 대해 서술하고자 한다.

### 12.2 Gantt Chart

Activity	1주	2주	3주	4주	5주	6주	7주	8주	9주	10주	11주	12주	13주	14주	15주
Idea	■	■													
Feasibility Study			■	■											
Requirement Analysis			■	■	■										
Design Study				■	■	■	■								
Testing						■	■	■	■	■					
Design							■	■	■						
Develop								■	■	■	■	■	■		
Integrate Testing											■	■	■	■	■
Document work				■	■	■	■	■	■	■	■	■	■	■	■

<Figure 12.1 Gantt chart>

위의 Gantt chart는 docking system 의 예상 진행 plan을 보여준다. 하지만 실제 일정은 예상과 바뀔수 있고, 실제 일정이 바뀌게 될 경우 차트가 갱신될 것이다.

# 13 Index

## 13.1 Table Index

<Table 9.1 Relational Schema of User>	78
<Table 9.2 Relational Schema of Dog>	78
<Table 9.3 Relational Schema of Walking>	79
<Table 9.4 Relational Schema of Contents>	79
<Table 9.5 Relational Schema of Comments>	80
<Table 9.6 Relational Schema of Tag>	80
<Table 9.7 Relational Schema of Follow>	81
<Table 9.8 data definition language form of the table User>	81
<Table 9.9 data definition language form of the table Dog>	82
<Table 9.10 data definition language form of the table Walking>	82
<Table 9.11 data definition language form of the table Contents>	83
<Table 9.12 data definition language form of the table Comments>	83
<Table 9.13 data definition language form of the table Tag>	83
<Table 9.14 data definition language form of the table Follow>	84

## 13.2 Figure Index

<Figure 2.1 UML>	10
<Figure 2.2 package diagram example>	11
<Figure 2.3 deployment diagram example>	12
<Figure 2.4 class diagram example>	13
<Figure 2.5 state diagram example>	13
<Figure 2.6 sequence diagram example>	14
<Figure 2.7 ER diagram example>	15
<Figure 2.8 drawio>	15
<Figure 2.9 Android Studio>	16
<Figure 2.10 OpenGL ES>	16
<Figure 2.11 Firebase>	16
<Figure 2.12 Amazon web Services>.	17
<Figure 2.13 Google Map API>	17
<Figure 2.14 OpenWeatherMap API>	17
<Figure 2.15 Arduino>	18
<Figure 8.1 JSON>	60
<Figure 9.1 Entity Symbol>	67
<Figure 9.2 Relationship Symbol>	68

<Figure 9.3 Attribute Symbol>	68
<Figure 9.4 Cardinality and ordinality of ER Diagram>	69
<Figure 11.1 Android Studio Symbol>	89
<Figure 11.2 Firebase Logo>	89
<Figure 11.3 Arduino Logo>	90
<Figure 11.4 Github Logo>	90
<Figure 12.1 Gantt chart>	92

### 13.3 Diagram Index

<Diagram 2.1 Community System>	18
<Diagram 2.2 User information System>	19
<Diagram 2.3 Matching System>	20
<Diagram 2.4 Peristalsis System>	20
<Diagram 3.1 System Organization Block Diagram>	21
<Diagram 3.2 Community System Architecture>	22
<Diagram 3.3 User Information System Architecture>	23
<Diagram 3.4 Matching System Architecture>.	24
<Diagram 3.5 Peristalsis System Architecture>	25
<Diagram 3.6 Context Diagram of Docking>	26
<Diagram 3.7 Use Case Diagram of Docking>	27
<Diagram 3.8 package Diagram of Docking>.	28
<Diagram 3.9 Deployment Diagram of Docking>	29
<Diagram 3.10 Docking ER Diagram>	30
<Diagram 3.11 Docking ER Diagram ver.2>	30
<Diagram 4.1 Class Diagram of Community System>	31
<Diagram 4.1 Community Main page Sequence Diagram>	34
<Diagram 4.2 Uploading Sequence Diagram>	35
<Diagram 4.3 Follow Sequence Diagram>	35
<Diagram 4.4 Tag Sequence Diagram>	36
<Diagram 4.5 Comments Sequence Diagram>	36
<Diagram 4.6 Searching Sequence Diagram>	37
<Diagram 4.7 Community main page State Diagram>	38
<Diagram 4.8 Uploading State Diagram>	39
<Diagram 4.9 Follow State Diagram>	40
<Diagram 4.10 Tag State Diagram>	41
<Diagram 4.11 Comments State Diagram>	42
<Diagram 4.12 Searching State Diagram>	43

<Diagram 5.1 Class Diagram of User Information System>	44
<Diagram 5.2 Sequence diagram of Sign up with Naver >	45
<Diagram 5.3 Sequence diagram of Sign up with Docking >	46
<Diagram 5.4 Sequence diagram of Log in with Naver >	46
<Diagram 5.5 Sequence diagram of Log in with Docking >	46
<Diagram 5.6 Sequence diagram of Dog List Management >	47
<Diagram 5.7 Sequence diagram of User Account Management >	47
<Diagram 5.8 State diagram of Sign in>	48
<Diagram 5.9 State diagram of Sign out>	49
<Diagram 5.10 State diagram of Log in>	50
<Diagram 5.11 State diagram of User Account Access>	51
<Diagram 6.1 Class diagram of Matching System>	52
<Diagram 6.2 Sequence diagram for matching system>	54
<Diagram 6.3 State diagram of Walking Info>	55
<Diagram 6.4 State diagram of Mate matching & DM>	56
<Diagram 7.1 Sequence Diagram of Band Sync>	57
<Diagram 7.2 State Diagram of Band>	58
<Diagram 7.3 Class Diagram of Band>	59
<Diagram 9.1 Overall ER Diagram ver1>	69
<Diagram 9.2 Overall ER Diagram ver2>	70
<Diagram 9.3 User Entity Diagram>	70
<Diagram 9.4 Dog Entity Diagram>	71
<Diagram 9.5 Walking Entity Diagram>	71
<Diagram 9.6 Contents Entity Diagram>	71
<Diagram 9.7 Comments Entity Diagram>	72
<Diagram 9.8 Tag-Contents Relation Diagram>	72
<Diagram 9.9 Follow Relation Diagram>	73
<Diagram 9.10 Raise Relation Diagram>	73
<Diagram 9.11 Take Relation Diagram>	74
<Diagram 9.12 Write-contents Relation Diagram>	75
<Diagram 9.13 Have-reply Relation Diagram>	75
<Diagram 9.14 Write-comments Relation Diagram>	76
<Diagram 9.15 Have-re-reply Relation Diagram>	76
<Diagram 9.16 Tag-Comments Relation Diagram>	76